sKizzo: a Suite to Evaluate and Certify QBFs

Marco Benedetti

Istituto per la Ricerca Scientifica e Tecnologica (IRST)
Via Sommarive 18, 38055 Povo, Trento, Italy
`benedetti@itc.it`

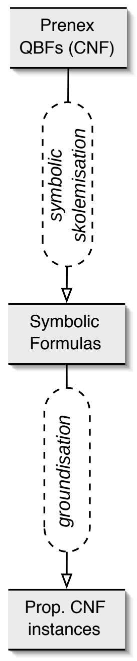
Abstract. We present `sKizzo`, a system designed to evaluate and certify Quantified Boolean Formulas (QBFs) by means of propositional skolemization and symbolic reasoning.

1 Introduction

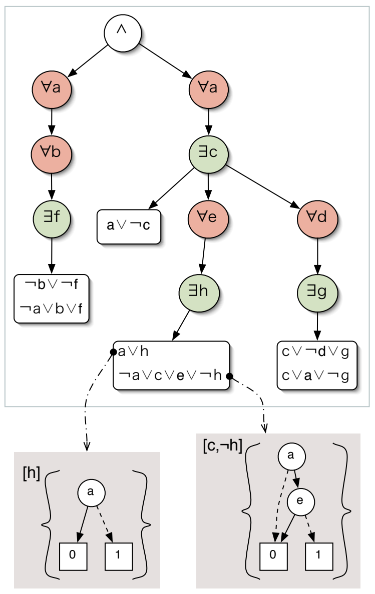
We present `sKizzo` [2,3], a software suite for dealing with *Quantified Boolean Formulas* (QBFs). `sKizzo` is mainly aimed at evaluating prenex CNF formulas by means of a novel *symbolic skolemization* technique[4]. In addition, it enables the user to (A) experiment with *quantifier trees*[6], (B) certify the (un)satisfiability of formulas[5] and (possibly) extract *unsatisfiable cores*, and (C) compute, manage, and query stand-alone certificates of satisfiability for QBFs. Both quantifier tree extraction and answer certification have never been attempted so far on QBFs.

At the hearth of `sKizzo` stays a new kind of symbolic representation for clauses and formulas, based on *Binary Decision Diagrams* (BDDs). As opposed to previous BDD-based approaches to propositional logic, `sKizzo`'s one employs a two-level data structure [2] designed to take advantage of the distinguishing features of QBFs. Besides allowing for a novel style of (complete/incomplete) *symbolic* reasoning, such representation makes it possible to unify within a coherent framework all the other approaches to QBF-satisfiability implemented so far. Namely: DPLL-like branching reasoning, q-resolution based algorithms, and compilation-to-SAT techniques.

2 Representation of QBF Instances and Data Structures



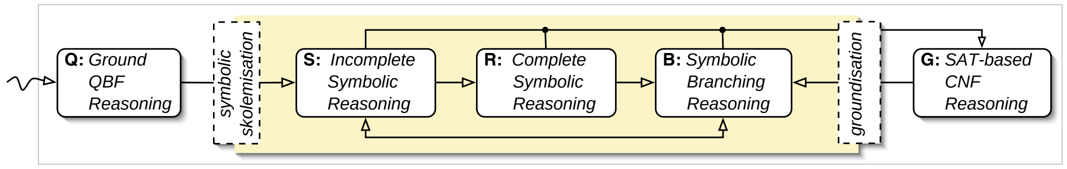
Three representation spaces for QBFs coexist within `sKizzo`. They are interconnected by two satisfiability-preserving trasformations (applied one-way), as reported in the picture aside. The first transformation leverages *outer skolemization* to map any (prenex CNF) instance $F \in QBFs$ onto a *symbolic* formula $\mathcal{F} = SymbSk(F)$, which is said to be *symbolic* as it couples list-based and BDD-based data structures to compactly represent a (possibly) exponentially less succinct propositional formula. The sentence $\mathcal{F}$ encodes the definability of a set of Skolem functions that capture a model (if any) of the original instance, according to the *symbolic skolemization* technique presented in [4]. A formal semantics is associated to symbolic formulas in such a way that $F \overset{sat}{\equiv} SymbSk(F)$ for every F. The other transformation—called *groundization*—translates a symbolic formula $\mathcal{F}$ into a purely existential CNF propositional instance $Prop(\mathcal{F})$ (a SAT problem) such that $F \overset{sat}{\equiv} SymbSk(F) \overset{sat}{\equiv} Prop(SymbSk(F))$. The role of these representations is as follows: Plain QBFs are handled in a pre-processing phase. Then, `sKizzo` moves to the symbolic representation and performs most of its work thereon. Zero or more CNF instances are generated/solved during the whole process (Section 2). Symbolic skolemization (and most of the processes described below) relies on the existence of a *quantifier tree* stating which existential variables are in the scope of which universal variables. Such tree-shaped structures are extracted out of the flat prenex input according to [6]. They replace linear prefixes so to more closely reflect the intrinsic dependencies in the matrix. A sample quantifier tree for the QBF $\forall a \forall b \exists c \forall d \forall e \exists f \exists g \exists h.(a \vee \neg c) \wedge (a \vee h) \wedge (c \vee \neg d \vee g) \wedge (\neg a \vee b \vee f) \wedge (\neg a \vee c \vee e \vee \neg h) \wedge (\neg b \vee \neg f) \wedge (a \vee c \vee \neg g)$ is depicted aside.



The symbolic representation is designed to allow for efficient forms of *symbolic reasoning* (Section 2), where universal reasoning is taken apart form existential reasoning (ROBDDs conveniently deal with the former, list-based representations with the latter). A symbolic formula is made up by symbolic clauses. During symbolic skolemization, one symbolic clause is extracted out of each QBF clause. The two major components of a symbolic clause $\Gamma_{\mathcal{I}}$ are a list Γ of existential literals and an index-set $\mathcal{I}$ represented via a ROBDD whose support set is the set of universal variables dominating the existential node at which the clause is attached in the quantifier tree. For example, the symbolic clauses $[h]_{\{00,01\}}$ and $[c, \neg h]_{\{10\}}$ are extracted out of $a \vee h$ and $\neg a \vee c \vee e \vee \neg h$ respectively (see the picture). Each symbolic clause $\Gamma_{\mathcal{I}}$ compactly represents a set $Prop(\Gamma_{\mathcal{I}})$ (with cardinality $|\mathcal{I}|$) of ground propositional clauses, in such a way that F is sat iff $Prop(\mathcal{F})$ is sat. For example, $Prop([c, g]_{\{01,10\}}) = \{c_0 \vee g_{01}, c_1 \vee g_{10}\}$. The *symbolic size* of $\mathcal{F}$ is $|\mathcal{F}|$, its *ground size* is $|Prop(\mathcal{F})|$: The initial symbolic size of $\mathcal{F}$ is thus linear in $|F|$. For details, see [4].

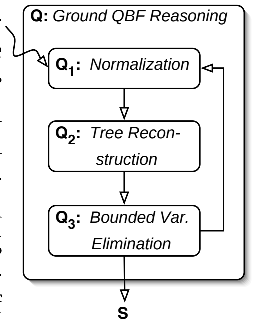
3 Inference Strategy

The *inference strategy* followed by sKizzo changes as the solution process goes on. Its evolution is described by a finite state machine whose inference states $S^{inf} = \{\mathsf{G}, \mathsf{S}, \mathsf{R}, \mathsf{B}, \mathsf{G}\}$ are traversed.


Q: Ground QBF Reasoning
symbolic skolemisation
S: Incomplete Symbolic Reasoning
R: Complete Symbolic Reasoning
B: Symbolic Branching Reasoning
groundisation
G: SAT-based CNF Reasoning

Each state in S^{inf} is associated to the application of an *inference style*. Each transition $x \rightarrow y$ in the picture, $x, y \in S^{inf}$, is labeled by a condition that triggers the shift from the style x to y (possibly requiring a satisfiability-preserving transformation).We now describe each state and transition.

Q: Ground QBF Reasoning. In the Q-state sKizzo works in the original QBF space, as represented aside. The step Q_1 amounts to apply the quantified form of three simple (incomplete) inference rules: *unit clause propagation*, *pure literal elimination*, and *forall-reduction*. The transition $\mathsf{Q}_1 \rightarrow \mathsf{Q}_2$ is triggered when all these rules reach their fixpoint. Bounded variable elimination (Q_3) applies *q-resolution* to eliminate a selected existentially quantified variable v in the deepest existential scope of some branch of the quantifier tree. This is done by substituting all the clauses containing v with the set of all the resolvents over v. As repeated applications of variable elimination often lead to an unmanageable explosion of the number of clauses, a *bounded* form of elimination is employed: Only variables whose elimination shrink the overall number of literals or clauses are eligible for elimination. The transition $\mathsf{Q}_3 \rightarrow \mathsf{Q}_1$ is selected when at least one variable has been eliminated during the last round, $\mathsf{Q}_3 \rightarrow \mathsf{S}$ is followed otherwise.



S: Incomplete Symbolic Reasoning. The instance is attacked by means of a set of (incomplete) *symbolic inference rules*, designed after their ground counterparts to achieve in one single application on symbolic clauses the same result they would obtain if applied to each ground clause separately.

SUCP (Symbolic Unit Clause Propagation). This rule builds on top of the observation that each symbolic unit clause $[\gamma]_{\mathcal{I}}$ in the formula represents a set $\{\gamma_i | i \in \mathcal{I}\}$ of ground unit literals. All of them are symbolically assigned at once to avoid an immediate contradiction.

SPLE (Symbolic Pure Literal Elimination). This rule computes a symbolic representation for the set of pure literals, then simplify the formula by assigning all of them at once. It comes in two flavors: a monolitic (one variable per step) and an incremental (one clause per step) version.

SSUB (Symbolic SUBsumption). This rules removes all the symbolic clauses that are subsumed by other clauses (*forward subsumption*). It employs scheduling heuristics, lazy computations, and a signature-based mechanism to minimize the overall effort. This rule complements the *backward subsumption* mechanism which is applied on-the-fly at each clause insertion.

SHBR (Symbolic Hyper Binary Resolution). This rules enumerates all the resolution chains of binary symbolic clauses in the formula, looking for contradictions. Each such contradiction determines a necessary consequence of the formula, compactly represented as a unit symbolic clause which is added to the instance (SUCP then draws all the entailed consequences).

SER (Symbolic Equivalency Reasoning). This rules look for non-empty strongly connected components in the *symbolic binary implication graph*[2] of the formula. Each such component determines a symbolic equivalence which is applied to simplify the formula.

A carefully designed application schedule is necessary to profit from the above set of rules as a whole. `sKizzo` implements a dynamic scheduling policy which works as follows.

1. The inference process is divided into subsequent *inference rounds*. At each round, the rules that have the rights to do so (see below) are sequentially executed.
2. The rule currently working is monitored during its execution. When certain resource limits are exceeded (inference steps undertaken, time elapsed, memory allocated, etc.), the rule is preemptively stopped (the rule's context is saved to re-start working from the interruption point).
3. When all the rules in the inference round have been executed, they are ranked according to their *relative efficiency*. The resource limits for the next rounds are re-distributed on a meritocratic basis: the better a rule has proved to be, the larger the resources it will be granted next.
4. In addition, rules failing to be effective loose the right to execute for a number of inference steps that enlarges with the number of rounds they have been performing poorly. The longer they keep on being ineffective, the more sparingly they are given a try.

The assessment of rules' efficiency is a major issue in the above policy. As all the rules reduce the ground size of the instance at each application (conversely, the symbolic size might be enlarged), the ground-size-shrink-percentage-per-resource-unit is assumed as a measure of efficiency. This measure needs itself resources to be computed. When BDD primitives and lazy evaluation do not suffice to keep the cost of assessment within pre-established limits, `sKizzo` resorts to approximated measures. The transition S $\rightarrow$ G is triggered if the ground size of the current problem becomes *affordable* via SAT-based reasoning (see the G-style), unless the symbolic reasoning is behaving so efficiently that ground reasoning is estimated not to pay back. The transition S $\rightarrow$ R is activated when the rules adopted come out to be unable to solve the problem. This happens under two circumstances: (1) the overall fixpoint is reached but no decision is obtained, or (2) the rate at which the problem is being shrunk has been staying below a certain threshold since a given number of inference rounds.

R: Complete Symbolic Reasoning. This state is similar to S, with one major exception: a refutationally complete rule is inserted in the pool of symbolic rules exercised at each inference round.

SDR (Symbolic Directional Resolution). This rules eliminates one symbolic variable per step by substituting the set of resolving clauses with the set of their symbolically computed resolvents.

Efficiency as size-shrinking measurement is unfair for SDR. This rule may need to pass through intermediate clause-sets that are much larger than the originating instance to come to a solution. So, SDR is given the change to consume more and more resources regardless of the size of the formula it is constructing. The other rules are still applied/evaluated in a round robin way (SSUB is especially useful here to reduce the redundancy SDR generates). Two outcomes are possible: (1) the largest intermediate result fits within the physical memory of the machine on which sKizzo is running—so the instance is solved, or (2) an out-of-memory condition occurs. As `sKizzo` keeps on monitoring its own resource consumption, he is able to detect the latter occurrence and give up resolution-based reasoning. The transition $\mathsf{R} \rightarrow \mathsf{B}$ is triggered. As usual, the transition $\mathsf{R} \rightarrow \mathsf{G}$ is followed if (and as soon as) the current problem becomes *affordable* via SAT-based reasoning (see the G-style).

A checkpointing mechanism is implemented against the unlucky possibilities that no consistent formula representation exists when mem-out occurs, or the formula yielded by SDR is so larger than the input formula that we would prefer to restart working on the original version: Symbolic formulas have to be explicitly committed or rolled-back depending on their eventual characteristics. This ensures that blow-up phenomena do not negatively affect the rest of the inference process.

B: Branching Reasoning. In this status, a search-based branching decision procedures extending the DPLL approach to the quantified case is applied. Models are searched following the left-to-right prefix order of variables during a depth-first visit of the semantic evaluation tree of the formula. Existential variables generate *or* nodes that disjunctively split the branch, universal quantifiers are associated to *and* nodes that split branches conjunctively. Distinguishing features of `sKizzo`:

– Both universal and existential splits are performed symbolically.
– The partial order induced by the internal structure of the quantifier tree is substituted for the left-to-right order of variables in the prefix. The main advantage is that nodes with more than one child induce sets of *disjoint* sub-instances that are solved in isolation of one another.
– After each existential split, the cofactored matrix undergoes further incomplete symbolic normalization (transition $\mathsf{B} \rightarrow \mathsf{S}$ and back). This mechanism extends the unit-clause-propagation based form of look-ahead used in purely branching solvers.
– The base case of the recursion does not deal with trivial sub-formulas. Well in advance, either symbolic reasoning (transition $\mathsf{B} \rightarrow \mathsf{S}$, whenever the current instance falls within its deductive power) or ground reasoning (transition $\mathsf{B} \rightarrow \mathsf{G}$, whenever the ground version of the problem is affordable) decide every sub-instance, acting as powerful look-ahead tools.

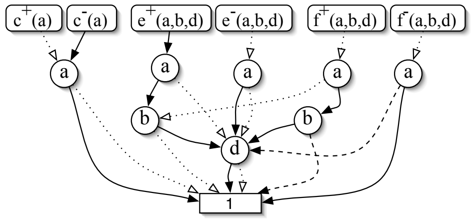
Many enhancements to the basic DPLL procedure are implemented. A conflict-analysis machinery is employed in the event of inconsistent partial assignment to isolate the branching steps responsible for the contradiction to arise. This information is used to perform a conflict-directed backjumping. As contradictions follow in general from a mix of branching steps, symbolic reasoning, and SAT-based reasoning, the three of these inference styles share a common conflict-analysis engine. A symbolic learning mechanism extracts symbolic clauses out of contradictions (useful to prune the rest of the search). Size-bounded and relevance-bounded heuristics are used to constraint the required amount of memory. Branching heuristics are also enrolled: MOMS and VSDIS are implemented.

G: SAT-based Ground Reasoning. In the G-state we explicitly construct $Prop(SymbSk(F))$ and solve it via state-of-the-art SAT solvers (they come out to be very efficient on such instances). This amounts to (1) build an encoding from the structured namespace of symbolic literals/clauses onto a flat propositional space, (2) generate all the necessary clauses, (3) make the SAT solver handle the resulting instance: Quite some "almost-existential" families of instances are successfully dealt with in the G status (hash-table based mechanism are implemented to make the translation fast).

A transition $x \rightarrow \mathsf{G}$, $x \in \{\mathsf{S}, \mathsf{R}, \mathsf{B}\}$, is triggered as soon as the groundization of the current formula becomes *affordable*. At the beginning, this notion is simply given in terms of memory requirements: The ground version of the instance fits into the memory and leaves enough space for the SAT solver to work. By construction, the transitions $x \rightarrow \mathsf{G}$, $x \in \{\mathsf{S}, \mathsf{R}\}$ are triggered at most once, yielding an instance SAT-equivalent to the original QBF problem. Conversely, B generates a (possibly) long chain of SAT instances, each one encoding the outcome of the exploration of an entire sub-tree of the QBF semantic evaluation tree. Along this chain, the notion of affordability is adjusted by a learning algorithm that tries to guess the optimal switch size between B and G. Furthermore, for the G-state to actively participate in conflict analysis, we map unsatisfiable ground cores (extracted by analyzing the ground inference trace) onto symbolic cores, then onto branching choices.

4 Certification

`sKizzo` implements a mechanism to certify its claims of (un)satisfiability. Evaluation and certification are completely decoupled, with almost no overhead for the former. The two meshes of the chain are connected through an *inference log*, produced by the solver and subsequently red by an external *certifier*. The log contains information about (1) the context switches between inference styles, (2) the sequence of the (symbolic) instantiations of inference rules undertaken (resolutions, substitutions, assignments), (3) entries for rollback/commit points and other control information.



By reading the log forward, the certifier is able to reproduce the derivation of the empty clause (unsat instances) and its graph of dependencies, thus extracting an unsatisfiable core. On sat instances, the certifier applies an *inductive model reconstruction*[5] procedure while parsing the log backward. It constructs a stand-alone, BDD-based sat-certificate encoding a QBF model. As an example, the picture aside depicts the sat-certificate produced for $\forall a \forall b \exists c \forall d \exists e \exists f.\ (\neg b \vee e \vee f) \wedge\ (a \vee c \vee f) \wedge\ (a \vee d \vee e) \wedge\ (\neg a \vee \neg b \vee \neg d \vee e) \wedge\ (\neg a \vee b \vee \neg c) \wedge\ (\neg a \vee \neg c \vee \neg f) \wedge\ (a \vee \neg d \vee \neg e) \wedge\ (\neg a \vee d \vee \neg e) \wedge\ (a \vee \neg e \vee \neg f)$. It is possible to verify that a model is encoded into such certificate: By assigning the existential variable e (similarly for c and f) to TRUE when $e^+(a,b,d) = 1$ and to FALSE when $e^-(a,b,d) = 1$ the matrix is always satisfied.

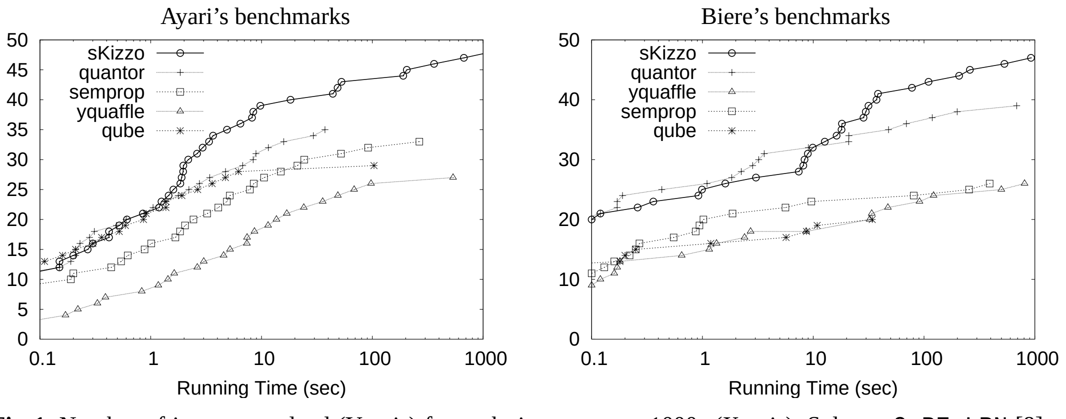


Fig. 1. Number of instances solved (Y axis) for each timeout up to 1000s (X axis). Solvers: `QuBE-LRN` [8], v. 1.3, a search-based solver featuring lazy data structures for unit clause and pure literal propagation, plus conflict and solution learning. `Quantor` [7], v. 2004.01.25, a solver employing q-resolution and expansion to eliminate quantifiers, plus optimizations to improve efficiency. `SEMPROP` [10], v. 24.02.02, a search-based solver featuring directed backtracking and lemma/model caching. `yQuaffle` [11], v. 09.30.04, a search-based solver featuring multiple conflict-driven learning, inversion of quantifiers and solution-based backtracking.

Instance	$\forall$	$\exists$	*Al.*	T_s	*Instance*	$\forall$	$\exists$	*Al.*	T_s
adder12	942	1722	3	190.0	*cnt15*	15	1457	30	33.2
adder14	1281	2359	3	670.0	*cnt16*	16	1650	32	44.7
adder16	1672	3096	3	1200.0	*s713_d3_s*	791	3098	2	384.7
cnt09re	9	609	18	259.6	*s499_d7_s*	896	4039	2	107.5
cnt10r	10	704	20	67.5	*s499_d10_s*	1355	5971	2	493.1
cnt10e	10	704	20	923.1	*s386_d3_s*	312	1467	2	23.9
cnt11r	11	840	22	190.38	*s386_d4_s*	471	2118	2	631.0
cnt12r	12	988	24	548.68	*s386_d5_s*	630	2769	2	795.3

instance	$\forall$	$\exists$	*Al.*	T_s	T_r	T_v	$\|\mathcal{L}\|$	$\|\mathcal{C}\|$
adder-16	1672	3096	3	1200.0	2025.2	1.1	$3.5{\cdot}10^3$	$2.4{\cdot}10^5$
Adder2-10	545	7424	5	360.0	97.8	0.1	$8.5{\cdot}10^3$	$6.1{\cdot}10^4$
cnt09re	9	609	18	280.0	0.4	0.1	$5.8{\cdot}10^3$	$9.0{\cdot}10^1$
cnt16	16	1650	32	45.0	36.0	0.1	$3.4{\cdot}10^5$	$5.9{\cdot}10^2$
k_grz_n18	24	767	16	55.0	0.8	0.1	$1.8{\cdot}10^3$	$3.2{\cdot}10^3$
k_poly_n18	110	1354	112	4.8	11.6	0.1	$4.5{\cdot}10^3$	$1.1{\cdot}10^3$
k_ph_n15	10	4833	4	86.0	1.5	0.4	$1.1{\cdot}10^4$	$2.8{\cdot}10^3$
k_d4_n16	69	1368	40	11.0	149.0	0.3	$1.1{\cdot}10^4$	$4.8{\cdot}10^4$

Table 1. On the left: some *2004-hard* instances (i.e. remained unsolved during the SAT04 evaluation of QBF solvers) solved by `sKizzo`. On the right: evaluation compared to SAT-certification. We report: the number of existential ($\exists$) and universal ($\forall$) variables, the number of quantifier alternations ($Al.$), the time taken to solve/reconstruct/verify (T_s,T_r,T_v), the number $|\mathcal{L}|$ of steps in the log and of nodes $|\mathcal{C}|$ in the certificate.

5 Implementation and experimentation

`sKizzo` is a 50k-line piece of code written in C using an object-oriented programming style. It has been developed on a PowerPC/MacOS X platform, then migrated to i386/Linux. It relies on the CUDD package 2.4.0 and DDDMP 2.0 for BDD manipulations, and on zChaff 2004.5.13 and siege v4 for SAT solving. Command-line options allow the user to individually (de)activate inference rules, and to construct *solving personalities* by forbidding the visit of some states of the inference FSM. Syntactic trees, CNF instances and certificates may be dumped to secondary memory.

The experimental evaluation of our suite involves the assessment of the relative strengths of different solving personalities and an analysis of how certification performances relate to solving performances. This yields a large amount of data, for which we refer the reader to [3].

Here we limit our presentation to a performance comparison (shown in Figure 1 and performed on a 2.6 GHz P4, 1Gb main memory, running Linux v2.4) between `sKizzo` and the best publically available state-of-the-art QBF solvers [9] over two challenging groups of QBF instances extracted from the QBFLIB's archive [8]: Biere's benchmarks [7], made up of 64 instances divided into 4 families, where the n-th instance in each family refers to a model checking problem on a n-bit counter, and Ayari's benchmarks [1], made up of 72 instances divided into 5 families, obtained from real-world verification problems on circuits and protocol descriptions.

Table 1 presents a few more results showing that `sKizzo` has solved instances never solved before. In addition, the results concerning SAT-certificate extraction suggest that certification is actually feasible. Although QBF verification cannot be in general accomplished in polynomial time, the task of building and verifying a certificate comes out not to be overwhelming on application-related instances: `sKizzo` has been able to certify all the satisfiable formulas it has solved. Also, memory requirements for certificates are within the capabilities of current machines.

References

1. A. Ayari and D. Basin. Bounded Model Construction for Monadic Second-order Logics. In *Proc. of CAV'00*, 2000.
2. M. Benedetti. sKizzo: a QBF Decision Procedure based on Propositional Skolemization and Symbolic Reasoning, Tech.Rep. 04-11-03, ITC-irst, 2004.
3. M. Benedetti. `sKizzo`'s web site, `sra.itc.it/people/benedetti/sKizzo`, 2004.
4. M. Benedetti. Evaluating QBFs via Symbolic Skolemization. In *Proc. of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR04)*, number 3452 in LNCS. Springer, 2005.
5. M. Benedetti. Extracting Certificates from Quantified Boolean Formulas. In *Proc. of IJCAI05*, 2005.
6. M. Benedetti. Quantifier Trees for QBFs. In *Proc. of SAT05*, 2005.
7. A. Biere. Resolve and Expand. In *Proc. of SAT'04*, pages 238–246, 2004.
8. E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE: A system for deciding Quantified Boolean Formulas Satisfiability. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001)*, 2001.
9. D. Le Berre, M. Narizzano, L. Simon, and A. Tacchella. Second QBF solvers evaluation, avaliable on-line at `www.qbflib.org`, 2004.
10. R. Letz. Advances in Decision Procedures for Quantified Boolean Formulas. In *Proceedings of the First International Workshop on Quantified Boolean Formulae (QBF'01)*, pages 55–64, 2001.
11. L. Zhang and S. Malik. Towards Symmetric Treatment of Conflicts And Satisfaction in Quantified Boolean Satisfiability Solver. In *Proc. of CP'02*, 2002.