Resolve and Expand

Armin Biere

Johannes Kepler University
Institute for Formal Models and Verification
Altenbergerstrasse 69, A-4040 Linz, Austria
biere@jku.at

Abstract. We present a novel expansion based decision procedure for quantified boolean formulas (QBF) in conjunctive normal form (CNF). The basic idea is to resolve existentially quantified variables and eliminate universal variables by expansion. This process is continued until the formula becomes propositional and can be solved by any SAT solver. On structured problems our implementation **quantor** is competitive with state-of-the-art QBF solvers based on DPLL. It is orders of magnitude faster on certain hard to solve instances.

1 Introduction

Recent years witnessed huge improvements in techniques for checking satisfiability of propositional logic (SAT). The advancements are driven by better algorithms on one side and by new applications on the other side. The logic of quantified boolean formulas (QBF) is obtained from propositional logic by adding quantifiers over boolean variables. QBF allows to represent a much larger class of problems succinctly.

The added expressibility unfortunately renders the decision problem PSPACE complete [20]. Nevertheless, various attempts have been made to lift SAT technology to QBF, in order to repeat the success of SAT. The goal is to make QBF solvers a versatile tool for solving important practical problems such as symbolic model checking [13] or other PSPACE complete problems.

For QBF the nesting order of variables has to be respected. Accordingly two approaches to solve QBF exist. Either variables are eliminated in the direction from the outermost quantifier to the innermost quantifier or vice versa. We call the first approach *top-down*, and the second one *bottom-up*.

Current state-of-the-art QBF solvers [4,17,12,9,22] are all top-down and implement a variant of the search-based Davis & Putnam procedure DPLL [7]. Additionally, QBF requires decision variables to be chosen in accordance with the quantifier prefix. Learning has to be adapted to cache satisfiable existential sub goals. DPLL also forms the basis of most state-of-the-art SAT solvers, and therefore it was most natural to use it for QBF as well.

Even for SAT, there are alternatives to DPLL, based on variable elimination, such as the resolution based Davis & Putnam procedure DP [8]. It has never been used much in practice, with the exception of [5], since usually too many clauses are generated.

Eliminating variables by resolution as in DP can be lifted from SAT to QBF as well. The result is a bottom-up approach for QBF called Q-resolution [10]. The only difference between Q-resolution and ordinary resolution is, that in certain cases universally quantified variables can be dropped from the resolvent.

In theory, Q-resolution is complete but impractical for the same reasons as resolution based DP [8]. It has not been combined with compact data structures either. In our approach, we apply Q-resolution to eliminate innermost existentially quantified variables. To make this practical, we carefully monitor resource usage, always pick the cheapest variable to eliminate, and invoke Q-resolution only if the size of the resulting formula does not increase much. We use *expansion* of universally quantified variables otherwise.

Expansion of quantifiers has been applied to QBF in [1] and used for model checking in [21,2]. All three approaches work on formulae or circuit structure instead of (quantified) CNF. We argue that CNF helps to speed up certain computationally intensive tasks, such as the dynamic computation of the elimination schedule. First it is not clear how Q-resolution can be combined with this kind of structural expansion. In addition our goal is to eventually combine bottom-up and top-down approaches. CNF currently is the most efficient data structure for representing formulas in top-down approaches for SAT.

Another general bottom-up approach [16,14,15,6] is also based on quantifier elimination. A SAT solver is used to eliminate multiple innermost variables in parallel. In practice these approaches have only been applied to SAT or model checking. In principle it would be possible to apply them directly to QBF. In our approach single variables are eliminated one after the other. We can also alternate between either eliminating existential variables of the innermost scope and eliminating universal variables of the enclosing universal scope.

2 Preliminaries

Given a set of variables V, a literal l over V is either a variable v or its negation $\neg v$. A *clause* is a disjunction of literals, also represented by the set of its literals. A conjunctive normal form (CNF) is a conjunction of clauses. Assume that the set of variables is partitioned into m non empty scopes $S_1, \dots S_m, \subseteq V$, with $V = S_1 \cup \dots \cup S_m$ and $S_i \cap S_j = \emptyset$ for $i \neq j$. Each variable $v \in V$ belongs to exactly one scope $\sigma(v)$. Scopes are ordered linearly $S_1 < S_2 \dots < S_m$, with S_1 the outermost and S_m the innermost scope. For each clause C the maximal scope $\sigma(v)$ over all variables v in C is unique and defined as the scope $\sigma(C)$ of C. The scope order induces a pre-order on the variables which we extend to an arbitrary linear variable order.

Each scope is labelled as *universal* or *existential* by the labelling $\Omega(S_i) \in \{\exists, \forall\}$. Variables are labelled with the label of their scope as $\Omega(v) \equiv \Omega(\sigma(v))$. We further require that the ordered partition of V into scopes is maximal with respect to the labelling, or more precisely $\Omega(S_i) \neq \Omega(S_{i+1})$ for $1 \leq i < m$.

Now a quantified boolean formula (QBF) in CNF is defined as a CNF formula f together with an ordered partition of the variables into scopes. This definition matches the QDIMACS formats [11] very closely, with the additional restriction of maximality.

A variable v is defined to occur in positive (negative) phase, or just positively (negatively), in a clause C, if C contains the literal v ($\neg v$). A clause in which a variable occurs in both phases is *trivial* and can be removed from the CNF. Two clauses C, D, where v occurs positively in C and negatively in D, can be resolved to a resolvent clause. The resolvent consists of all literals from C except v and all literals from D except $\neg v$.

For a non-trivial clause C we define the process of *forall reduction* as follows. The set of *forall reducible variables* in C is defined as the set of universal variables in C for which there is no larger existential variable in C, with respect to the variable order. The clause D obtained from C by forall reduction contains all variables of C except forall reducible variables. For instance the two clauses in the following QBF

$$\exists x \,.\, \forall y \,.\, (x \vee y) \wedge (\neg x \vee \neg y)$$

are *not* forall reduced. Forall reduction results in removing the literal y in the first clauses and the literal $\neg y$ in the second, which results in two contradicting units. Also note, that forall reduction can result in an empty clause if the original clause contains universal variables only. Plain resolution followed by forall reduction is the same as Q-resolution [10].

Forall reduction is an equivalence preserving transformation. Thus without loss of generality we can assume that the CNF is in forall reduced form: by forall reduction no clause can be reduced further. This assumption establishes the important invariant, that $\Omega(\sigma(C)) = \exists$ for all clauses C. In other words, all clauses have an existential scope. There are no clauses with a universal scope. Particularly, the innermost scope is always existential ($\Omega(S_m) = \exists$). In our implementation, for each existential scope, we maintain a list of its clauses, and for each clause a reference to its scope.

3 Elimination

We eliminate variables until the formula is propositional and contains only existential quantifiers. Then it can be handed to a SAT solver. After establishing the invariant discussed above, a non-propositional QBF formula has the following structure

$$\Omega(S_1)\, S_1 \,.\, \Omega(S_2)\, S_2 \,.\, \dots \forall S_{m-1} \,.\, \exists\, S_m \,.\, f \,\wedge\, g \qquad m \geq 2 \tag{1}$$

where the formula f is *exactly* the conjunction of clauses with scope S_m. We either eliminate a variable in the innermost existential scope $S_\exists \equiv S_m$ by Q-resolution or a variable in the innermost universal scope $S_\forall \equiv S_{m-1}$ by expansion.

3.1 Resolve

An existential variable v of $S_\exists$ is eliminated as in [8,10] by performing all resolutions on v, adding the forall reduced resolvents to the CNF, and removing all clauses containing v in either phase. As example consider the clauses in Fig. 1.

We assume that these 7 clauses are all clauses of a CNF in which the innermost existential variable v occurs. To eliminate v, we simply perform all 3×2 resolutions between a clause on the left side, in which v occurs positively, with all clauses on the

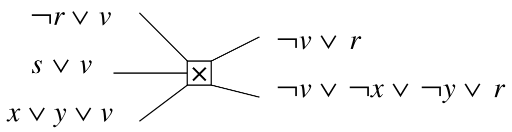

Fig. 1. Number of resolution pairs is quadratic.

right side, in which v occurs negatively. In this case 3 resolvents are trivial. The other three resolvents

$$(s \vee r), \qquad (x \vee y \vee r), \quad \text{and} \quad (s \vee \neg x \vee \neg y \vee r)$$

are added to the CNF and the original 5 clauses containing v in either phase are removed. As always, before adding one of the clauses, forall reduction is applied.

3.2 Expand

Expansion of a universal variable v in $S_\forall$ requires to generate a copy $S'_\exists$ of $S_\exists$, with a one-to-one mapping of variables $u \in S_\exists$ mapped to $u' \in S'_\exists$. With f' we denote the conjunction of clauses obtained from f by replacing all occurrences of $u \in S_\exists$ by u'. The result of expanding $v \in S_\forall$ in Eqn. (1) is as follows

$$\Omega(S_1)\, S_1 \,.\, \Omega(S_2)\, S_2 \,.\, \ldots \forall(S_\forall - \{v\}) \,.\, \exists(S_\exists \cup S'_\exists) \,.\, f\{v/0\} \wedge f'\{v/1\} \wedge g$$

By $f\{v/0\}$ we denote the result of substituting v by the constant 0 in f. This is equivalent to removing all clauses in which v occurs in negative phase and removing the occurrences of v in those clauses in which v occurs positively, followed by forall reduction. The substitution by 1 is defined accordingly.

4 Optimizations

Before invoking one of the two costly elimination procedures described in Sec. 3, we first apply unit propagation, a simple form of equivalence reasoning, and the standard QBF version of the pure literal rule. These simplifications are repeated until saturation.

4.1 Equivalence Reasoning

To detect equivalences we search for pairs of dual binary clauses. A clause is called *dual* to another clause if it consists of the negation of the literals of its dual. If such a pair is found, we take one of the clauses and substitute the larger literal by the negation of the smaller one throughout the whole CNF.

The search for dual clauses can be implemented efficiently by hashing binary clauses. In more detail, whenever a binary clause is added, we also save a reference to it in a hash table and check, whether the hash table already contains a reference to its dual. If this is the case an equivalence is found. After an equivalence is found, it is used to eliminate one of the variables of the equivalence. Consider the following QBF formula:

$$\exists x \,.\, \forall y \,.\, \exists z \,.\, \underline{(x \vee z)} \wedge (x \vee y \vee \neg z) \wedge (\neg x \vee \neg z) \wedge \underline{(\neg x \vee \neg z)}$$

The two underlined dual binary clauses involving x and z form an equivalence. After the last clause is added, the equivalence $x = \neg z$ is detected and z is replaced by $\neg x$, which results in the following QBF formula:

$$\exists x \,.\, \forall y \,.\, (x \vee \neg x) \wedge \underline{(x \vee y \vee x)} \wedge (\neg x \vee x) \wedge (\neg x \vee x)$$

After removal of 3 trivial clauses and forall reduction of the underlined clause, the only clause left is the unit clause x. In general, before searching for dual clauses, forall reduction has to be applied first. This way all substitutions triggered by equivalences will always replace existential variables by smaller literals. Replacing universal variables would be incorrect as the standard example $\exists x \,.\, \forall y \,.\, (x \vee \neg y) \wedge (\neg x \vee y)$ shows.

4.2 Subsumption

Expansion often needs to copy almost all clauses of the CNF. Moreover, the elimination procedures of Sec. 3 produce a lot of redundant subsumed clauses. Therefore, subsumed clauses should be removed. If a new clause is added, all old clauses are checked for being subsumed by this new clause. This check is called backward subsumption [19] and can be implemented efficiently on-the-fly, by using a signature-based algorithm. However, the dual check of forward subsumption [19] is very expensive and is only invoked periodically, for instance at each expansion step.

The subsumption algorithm is based on signatures, where a signature is a subset of a finite signature domain D. In our implementation $D = \{0, \ldots, 31\}$ and a signature is represented by an unsigned 32-bit word. Each literal l is hashed to $h(l) \in D$. The signature $\sigma(C)$ of a clause C is the union of the hash values of its literals. Finally, the signature $\sigma(l)$ of a literal l is defined as the union of the signatures of the clauses in which it occurs, and is updated whenever a clause is added to the CNF.

Let C be a new clause, which is supposed to be added to the CNF. Further assume that the current CNF already contains a clause D which is subsumed by C, or more formally $C \subseteq D$. Then the signature of C is a subset of the signature of D, which in turn is a subset of the signatures of all the literals in D. Since all the literals of C are also literals of D, we obtain the necessary condition, $\sigma(C) \subseteq \sigma(l)$ for all literals $l \in C$. The signature $\sigma(l)$ is still calculated with respect to the current CNF, to which C has not been added yet.

If this necessary condition fails, then no clause in the current CNF can be backward subsumed by the new clause. In this case our caching scheme using signatures is successful and we call it a cache hit. Otherwise, in the case of a cache miss, we need to traverse all clauses D of an arbitrary literal in the new clause, and explicitly check for $C \subseteq D$. To minimize the number of visited clauses, we take the literal with the smallest number of occurrences. During the traversal, inclusion of signatures is a necessary condition again. This can easily be checked, since the signature of a clause is constant, and can be saved.

In practice, the overhead of maintaining signatures and checking for backward subsumption in the way just described turns out to be low. For forward subsumption no such efficient solution exists, and thus, forward subsumption, in our implementation, is only invoked before expensive operations, like expansion. Then we remove all clauses, flush signatures and add back the clauses in reverse chronological order.

Finally, if a clause C is added to the CNF, the signatures of all its literals $l \in C$ have to be updated. However, if a clause is removed, hash collision does not allow to subtract its signature from all the signatures of its literals. Therefore we just keep the old signatures as an over approximation instead. After a certain number of clauses are removed a recalculation of accurate clause signatures is triggered.

4.3 Tree-Like Prefix

We also realized that there are situations in which a linear quantifier prefix is not optimal and the basic expansion step as described above copies too many clauses. Consider the QBF

$$\exists x \,.\, \forall y, u \,.\, \exists z, v \,.\; f_1(x,y,z) \wedge f_2(x,u,v)$$

It is a linearization of the following formula with a tree-like prefix:

$$\begin{array}{ccc} & \exists x & \\ & \wedge & \\ \forall y & & \forall u \\ \exists z & & \exists v \\ f_1(x,y,z) & & f_2(x,u,v) \end{array}$$

The result of expanding y as described above would contain redundant copies of clauses from f_2 and vice versa redundant copies of f_1 when expanding u. In general, this problem can be coped with in the copying phase of expansion. The idea is to copy only those clauses that contain a variable connected to the expanded variable. In this context we call a variable *locally connected* to another variable if both occur in the same clause. The relation *connected* is defined as the transitive closure of *locally connected*, ignoring variables smaller than the expanded variable and all other universal variables in the same scope.

This technique is cheap to implement and avoids to pay the price for one single expansion. But we have not found an efficient way to use the information about tree like scopes to generate better elimination schedules on-the-fly.

5 Scheduling

The remaining problem, and one of our key contributions, is an efficient algorithm for on-the-fly generation of elimination schedules. Our scheduler has to answer the question, which of the variables in $S_\exists \cup S_\forall$ to eliminate next. As a cost function for choosing the next variable we try to minimize the size of the CNF after elimination. The size is measured in number of literals, which is equal to the sum of sizes of all clauses. We separately calculate for each variable a pessimistic but tight upper bound on the number of literals added, if the variable is eliminated. The variable with the smallest bound, which can be negative, is chosen.

For each literal l we maintain two counters reflecting the number of occurrences $\mathrm{o}(l)$ and the sum $\mathrm{s}(l)$ of the sizes of the clauses in which l occurs. These counters need to be updated only when a clause is added or removed. The update is linear in the clause size. This also shows that the obvious alternative cost function, which minimizes the number of added clauses instead of literals, is less precise, without improving complexity. For each existential scope S we maintain a counter reflecting the sum $\mathrm{s}(S)$ of the sizes of its clauses.

5.1 Expansion Cost

For the expansion of $v \in S_\forall$ in Eqn. (1) according to Sec. 3.2 a tight upper bound on the number of added literals is calculated as follows. First f would be copied, which adds $\mathrm{s}(S_\exists)$ literals. In f clauses are removed in which v occurs negatively, in the copy f' clauses are removed in which v occurs positively. This means subtracting both $\mathrm{s}(v)$ and $\mathrm{s}(\neg v)$ from $\mathrm{s}(S_\exists)$. We also have to take care of the single literals removed, and the cost for eliminating v by expansion becomes

$$\mathrm{s}(S_\exists) \quad - \quad \big(\mathrm{s}(v) + \mathrm{s}(\neg v) + \mathrm{o}(v) + \mathrm{o}(\neg v)\big)$$

For all $v \in S_\forall$ the term $\mathrm{s}(S_\exists)$ is the same. Thus we only need to order these variables with respect to $-\big(\mathrm{s}(v) + \mathrm{s}(\neg v) + \mathrm{o}(v) + \mathrm{o}(\neg v)\big)$, which does not depend on other literals. This is essential for efficiency. In our implementation we use a separate heap based priority queue for each scope.

5.2 Resolving Cost

For the elimination of an existential variable $v \in S_\exists$ in Eqn. (1) according to Sec. 3.1 the calculation of a tight upper bound is similar but more involved. Consider Fig. 1. The literals on the left side, except v are copied $o(\neg v)$ times, which results in $o(\neg v) \cdot (s(v) - o(v))$ added literals. The number of copies of literals from the right side is calculated in the same way. Finally we have to remove all original literals, which all together results in the following cost, which again only depends on one variable:

$$\mathrm{o}(\neg v) \cdot \big(\mathrm{s}(v) - \mathrm{o}(v)\big) \quad + \quad \mathrm{o}(v) \cdot \big(\mathrm{s}(\neg v) - \mathrm{o}(\neg v)\big) \quad - \quad \big(\mathrm{s}(v) + \mathrm{s}(\neg v)\big)$$

As the example of Fig. 1 shows, this expression is only an upper bound on the cost of eliminating an existential variable by resolution. The bound is tight as the following example shows. Take the set of variables on each side of Fig. 1. If the intersection of these two sets only contain v, and all variables are existential, then the number of added literals exactly matches the bound.

Note that for bad choices of v calculating the multiplication may easily exceed the capacity of 32 bit integer arithmetic. Since variables with large costs can not be eliminated anyhow, we used saturating arithmetic with an explicit representation of infinity instead of arbitrary precision arithmetic.

5.3 Further Scheduling Heuristics

There are two exceptions to the scheduling heuristics just presented. First, as long as the minimal cost to eliminate an existential variable in $S_\exists$ is smaller than a given bound E, we eliminate the cheapest existential variable by resolution. This technique is also applied to pure propositional formulas. In this way **quantor** can be used as a preprocessor for SAT.

In our experiments, it turned out that in many cases, forcing the formula not to increase in size by setting $E = 0$, already reduces the final formula considerably. However, allowing small increases in size works even better. For scheduling purposes we use $E = 50$. This bound should probably be smaller if **quantor** is only used for preprocessing propositional formulas.

Another additional scheduling heuristics monitors the literals per clause ratio of the clauses with scope $S_\exists$. If it reaches a certain threshold, 4.0 in our implementation, an expansion is forced. After each forced expansion the threshold is increased by 10%. The reasoning behind forced expansion is as follows. A small literals per clause ratio increases the likelihood that the optimizations of Sec. 4 are applicable. In this sense, the scheduler should slightly bias decisions towards expansion instead of resolving, in particular, if the literals per clause ratio is high.

6 Experiments

We focus on structured instances, also called non-random, because we believe them to be more important for practical applications. As SAT solver we used **funex**, our own state-of-the-art SAT solver. It has not particularly been tuned towards our application. We have also seen rare cases where **funex** performs considerably worse than other SAT solvers, on SAT formulas generated by **quantor**.

In the first experiment we targeted the non random benchmarks of the SAT'03 evaluation of QBF [11] and compared **quantor** against **semprop** [12], the most efficient solver on these benchmarks in the evaluation [11]. We added **decide** [17] and **qube** with learning [9] as reference. In order to measure the effect of optimizations and using Q-resolution we also configured **quantor** in *expand* only mode. In this mode the scheduler always chooses expansion and all the optimizations are switched off. Exceptions are the pure literal rule, simplification by unit resolution, and forall reduction. This configuration, marked *expand* in Tab. 1, almost matches the original algorithm of the first version of **quantor**, which took part in SAT'03 evaluation of QBF [11].

As platform for this experiment we used an Intel Pentium IV 2.6 GHz with 1.5 GB main memory running Debian Linux. The results in Tab. 1 are clustered in families of benchmarks. For each family we count the number of instances solved in the given time limit of 32 seconds and memory limit of 1 GB. The numbers of families solved are printed in bold for best solvers. For a single best solver the numbers are underlined.

The comparison of the last two columns shows that expansion alone is very weak, and our new optimizations are essential to obtain an efficient state-of-the-art expansion based QBF solver. The number of cases in which **quantor** is among the best solvers for a family is the same as for **semprop**. There are four more families, for which **quantor**

benchmark family		#inst	**decide**	**qube**	**semprop**	*expand*	**quantor**
1	adder*	16	2	2	2	1	**3**
2	Adder2*	14	2	2	2	2	**3**
3	BLOCKS*	3	**3**	**3**	**3**	**3**	**3**
4	C[0-9]*	27	2	3	2	3	**4**
5	CHAIN*	11	10	7	**11**	4	**11**
6	comp*	5	4	4	**5**	**5**	**5**
7	flip*	7	6	**7**	**7**	**7**	**7**
8	impl*	16	12	**16**	**16**	**16**	**16**
9	k*	171	77	91	97	60	**108**
10	logn*	2	**2**	**2**	**2**	**2**	**2**
11	mutex*	2	1	**2**	**2**	**2**	**2**
12	qbf*	695	518	565	**694**	130	210
13	R3CNF*	27	**27**	**27**	**27**	25	21
14	robots*	48	0	**36**	**36**	15	24
15	term1*	4	2	**3**	**3**	1	**3**
16	toilet*	260	187	**260**	**260**	259	259
17	TOILET*	8	**8**	6	**8**	**8**	**8**
18	tree*	12	10	**12**	**12**	8	**12**
19	vonN*	2	**2**	**2**	2	**2**	**2**
20	z4ml*	13	**13**	**13**	**13**	**13**	**13**
#(among best in family)			**6**	**12**	**16**	**9**	**16**
#(single best in family)			**0**	**0**	**1**	**0**	**4**

Table 1. Number solved instances for benchmarks families of the QBF evaluation 2003.

is the single best solver, three more than for **semprop**. Also note, that the families qbf* and R3CNF*, on which **quantor** performs poorly compared to the other solvers, can actually be considered to be randomized.

A detailed analysis revealed that **quantor** was able to solve 10 instances classified as *hard* in [11]. These hard formulas could not be solved by any solver in 900 seconds during the SAT'03 evaluation of QBF [11]. In a second experiment we restricted the benchmark set to these hard instances, a far smaller set.

The new time limit was set to 800 seconds to accommodate for the slightly faster processor (2.6 GHz instead of 2.4 GHz in [11]). As predicted by the evaluation results in [11] all solvers except **quantor** timed out on these instances. The results for **quantor** are presented in Tab. 2. Only solved instances are listed and are not clustered into families, e.g. C49*1.*_0_0* is the single instance with file name matching this pattern.

In all but two of the cases where the full version of **quantor** succeeded the *expand* only version quickly reached the memory limit of 1 GB. We note the time until the memory limit was reached in parentheses. It is also remarkable that the memory requirements for **quantor** have a large variance. The columns $\forall$ and $\exists$ contain the number of universal quantifications by expansion and existential quantifications by resolution respectively.

	hard instance	*expand* time	*expand* space	*expand* ∀	**quantor** time	**quantor** space	**quantor** ∀	**quantor** ∃	**quantor** units	**quantor** pure	**quantor** subsu.	**quantor** subst.	**quantor** ∀red.
1	Adder2-6-s	(12.2)	m.o.	–	29.6	19.7	90	13732	126	13282	174081	0	37268
2	adder-4-sat	(12.1)	m.o.	–	0.2	2.8	42	1618	0	884	6487	0	960
3	adder-6-sat	(13.0)	m.o.	–	36.6	22.7	90	13926	0	7290	197091	0	54174
4	C49*1.*_0_0*	98.3	40.8	1	27.9	13.3	1	579	0	0	48	84	0
5	C5*1.*_0_0*	357.0	45.6	2	56.2	16.0	2	2288	10	0	4552	2494	0
6	k_path_n-15	(16.5)	m.o.	–	0.1	0.8	32	977	66	82	2369	2	547
7	k_path_n-16	(16.6)	m.o.	–	0.1	0.8	34	1042	69	85	2567	2	597
8	k_path_n-17	(16.2)	m.o.	–	0.1	0.9	36	1087	72	100	3020	2	639
9	k_path_n-18	(16.8)	m.o.	–	0.1	0.9	36	1146	76	106	3242	2	725
10	k_path_n-20	(21.4)	m.o.	–	0.1	0.9	38	1240	84	149	3967	2	855
11	k_path_n-21	(21.0)	m.o.	–	0.1	1.0	40	1318	84	130	4470	2	909
12	k_t4p_n-7	(16.8)	m.o.	–	15.5	105.8	43	88145	138	58674	760844	8	215
13	k_t4p_p-8	(21.4)	m.o.	–	5.8	178.6	29	12798	206	5012	85911	4	138
14	k_t4p_p-9	(21.2)	m.o.	–	0.3	4.5	32	4179	137	1389	23344	10	142
15	k_t4p_p-10	(17.3)	m.o.	–	27.9	152.9	35	130136	193	63876	938973	4	137
16	k_t4p_p-11	(17.3)	m.o.	–	86.0	471.5	38	196785	204	79547	1499430	4	140
17	k_t4p_p-15	(21.3)	m.o.	–	84.6	354.7	50	240892	169	181676	1336774	9	226
18	k_t4p_p-20	(20.9)	m.o.	–	3.6	16.1	65	27388	182	21306	197273	11	325

time in seconds, space in MB, m.o. = memory out (> 1 GB)

Table 2. Solved hard instances of SAT'03 evaluation of QBF.

We added columns containing the numbers of unit simplifications, applications of the pure literal rule, subsumed clauses, applied substitutions, and number of removed literals due to forall reduction (∀red). With the exception of subsumption, all optimizations are rather cheap with respect to run-time overhead, and as the data suggests, should be implemented. In particular the high number of pure literals in solving some instances is striking. Substitution does not seem to be important. More important, though also more costly, is subsumption.

For the two hard C[0-9]* instances covered in Tab. 2 more than 99% of the time was spent in the SAT solver. For the other solved hard instances no call to a SAT solver was needed. In an earlier experiment we used a slightly slower computer, an Alpha ES40 Server running at 666 MHz. The time limit was set to one hour, and the memory limit to 8 GB. In this setting, we were able to solve two more of the hard C[0-9]* benchmarks (with names matching C43*out*) in roughly 2500 seconds each. Again most time was spent in the SAT solver. Except for those reported in Tab. 2, no further hard instance of [11] could be solved within these limits.

We also like to report on experiments involving benchmarks from QBFLIB, which turned out to be very simple for **quantor**. These include two families of benchmarks consisting of the 10 impl* instances and the 14 tree* instances. These 24 instances can be solved altogether in less than 0.1 seconds.

One of the most appealing aspects of QBF is, that an efficient QBF solver may also be used for *unbounded* model checking via the translation of [18,20], also described in [17]. This translation needs only one copy of the transition relation but requires $2 \cdot l$

alternations of quantifiers, where $l = \lceil log_2 r \rceil$ and r is the initialized diameter (radius) of the model. In a boolean encoding l can be bounded by the number of state bits n. To check the hypothesis that QBF can be used for model checking in this way, we generated models of simple n-bit hardware counters, with reset and enable signal.

We check the invalid simple safety property, that the all-one state is not reachable from the initial state where all state bits are zero. This is the worst-case scenario for bounded model checking [3] since $2^n - 1$ steps are necessary to reach the state violating the safety property. Symbolic model checking [13] without iterative squaring needs 2^n fix point iterations. However, the size of the result of the translation of this problem to QBF is quadratic in n, the width of the counters.

With a time out of 60 seconds **decide** could only handle 3-bit-counters, **qube** and **semprop** up to 4 bits, while **quantor** solved 7 bits, matching the result by plain BMC with the same SAT solver. Since this example is very easy for BDD-based model checking, it is clear that QBF based model checking still needs a long way to go.

7 Conclusion

The basic idea of our QBF decision procedure is to resolve existential and expand universal variables. The key contribution is the resource-driven, pessimistic scheduler for dynamically choosing the elimination order. In combination with an efficient implementation of subsumption we obtain an efficient QBF solver for quantified CNF.

As future work we want to explore additional procedures for simplifying CNF and combine bottom-up elimination with top-down search. It may be also interesting to look into other representations, such as BDDs or ZBDDs.

Finally, we would like to thank Uwe Egly and Helmuth Veith for insisting on the argument that there is a benefit in not only focusing on a linear prefix normal form. Acknowledgements also go to Rainer Hähnle, whose comments triggered the optimization of our subsumption algorithm.

References

1. A. Ayari and D. Basin. QUBOS: deciding quantified boolean logic using propositional satisfiability solvers. In *Proc.* 4th *Intl. Conf. on Formal Methods in Computer-Aided Design (FMCAD'02)*, volume 2517 of *LNCS*. Springer, 2002.
2. P. Aziz Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT-solvers. In *Proc.* 6th *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*. Springer, 2000.
3. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proc.* 5th *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*. Springer, 1999.
4. M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proc. 16th National Conference on Artificial Intelligence (AAAI-98)*, 1998.
5. P. Chatalic and L. Simon. ZRes: The old Davis-Putnam procedure meets ZBDDs. In *17th Intl. Conf. on Automated Deduction (CADE'17)*, volume 1831 of *LNAI*, 2000.
6. P. Chauhan, E. M. Clarke, and D. Kröning. Using SAT based image computation for reachability analysis. Technical Report CMU-CS-03-151, Carnegie Mellon University, 2003.

7. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5, 1962.
8. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7, 1960.
9. E. Giunchiglia, M. Narizzano, and A. Tacchella. Learning for quantified boolean logic satisfiability. In *Proc. 18th National Conference on Artificial Intelligence (AAAI'02)*, 2002.
10. H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for quantified boolean formulas. *Information and Computation*, 117, 1995.
11. D. Le Berre, L. Simon, and A. Tacchella. Challenges in the QBF arena: the SAT'03 evaluation of QBF solvers. In *Proc. 6th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *LNCS*. Springer, 2003.
12. R. Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *Proc. Intl. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'02)*, volume 2381 of *LNCS*. Springer, 2002.
13. K. L. McMillan. *Symbolic Model Checking: An approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993.
14. K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Proc.* 14^{th} *Intl. Conf. on Computer-Aided Verification (CAV'02)*, volume 2404 of *LNCS*. Springer, July 2002.
15. M. Mneimneh and K. Sakallah. Computing vertex eccentricity in exponentially large graphs: QBF formulation and solution. In *Proc. 6th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *LNCS*. Springer, 2003.
16. D. Plaisted, A. Biere, and Y. Zhu. A satisfiability procedure for quantified boolean formulae. *Discrete Applied Mathematics*, 130(2), 2003.
17. J. Rintanen. Partial implicit unfolding in the Davis-Putnam procedure for quantified boolean formulae. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'01)*, 2001.
18. W. J. Savitch. Relation between nondeterministic and deterministic tape complexity. *Journal of Computer and System Sciences*, 4, 1970.
19. R. Sekar, I. V. Ramakrishnan, and A. Voronkov. Term indexing. In *Handbook of Automated Reasoning*, volume II. North-Holland, 2001.
20. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *5th Annual ACM Symposium on the Theory of Computing*, 1973.
21. P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining decision diagrams and SAT procedures for efficient symbolic model checking. In *Proc.* 12^{th} *Intl. Conf. on Computer Aided Conf. Verification (CAV'00)*, volume 1855 of *LNCS*. Springer, 2000.
22. L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *Proc. Intl. Conf. on Computer-Aided Design (ICCAD'02)*, 2002.