Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 174 (2007) 45–56

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Compressing BMC Encodings with QBF

Toni Jussila[1]   Armin Biere[2]

Institute for Formal Models and Verification
Johannes Kepler University, Linz, Austria

Abstract

Symbolic model checking is PSPACE complete. Since QBF is the standard PSPACE complete problem, it is most natural to encode symbolic model checking problems as QBF formulas and then use QBF decision procedures to solve them. We discuss alternative encodings for unbounded and bounded safety checking into SAT and QBF. One contribution is a linear encoding of simple path constraints, which usually are necessary to make k-induction complete. Our experimental results show that indeed a large reduction in the size of the generated formulas can be obtained. However, current QBF solvers seem not to be able to take advantage of these compact formulations. Despite these mostly negative results the availability of these benchmarks will help improve the state of the art of QBF solvers and make QBF based symbolic model checking a viable alternative.

Keywords: Bounded Model Checking, Encoding, QBF, SAT.

1 Introduction

Bounded Model Checking (BMC) [3] has the motivation to improve on BDD based symbolic model checking by using SAT procedures. Already in the original paper the use of QBF decisions procedures was suggested as a tool to make BMC complete without using BDDs. Completeness means that an LTL property can also be shown to hold as opposed to just being able to find counter examples. In this paper we focus on simple safety properties, for which we want to prove that a *bad* state is not reachable. More general properties can be handled for instance through techniques from [14].

The completeness result of [3] uses the fact that the diameter of the system, which is the length of the longest shortest path between two states, is an upper bound on the length of potential counter examples. The question is how the diameter can be calculated. In [3] a QBF formula is presented parameterized by d, which is satisfiable resp. *true*, iff d is an upper bound on the diameter. Nevertheless this

[1] Email: toni.jussila@jku.at

[2] Email: armin.biere@jku.at

doi:10.1016/j.entcs.2006.12.022

formulation was not used in the experiments, since efficient QBF solvers did not exist at that time.

In graph theory the notion of diameter is also known as *eccentricity*. To determine the eccentricity of the state transition graph of sequential circuits has been investigated in [10]. The authors used a dedicated QBF solver for quantifier elimination. However, the examples that could be handled are tiny. An argument why DPLL style QBF solvers can not handle this kind of problems well is given in [17]: in essence DPLL style QBF solvers need to perform an explicit state space search to determine the diameter.

However, it is possible to generate a purely propositional SAT formula without quantifiers for almost the same problem [3]. If it is unsatisfiable, it constitutes an upper bound on the diameter. This formula is parameterized by r and is unsatisfiable iff there is no *cycle free* path of length r. In graph terminology a cycle free path is also called *simple* path, while in [3] the maximal length of a simple path is called *reoccurrence diameter*.

These concepts can be refined in two ways [15]: first the diameter and the reoccurrence diameter can be *initialized*. The paths, both in the QBF and in the SAT case, can be forced to fulfill the additional constraint that exactly the first state is an initial state. Furthermore, instead of looking for maximal simple paths starting from an initial state in a *forward* manner, one can work *backward* from a bad state. In particular the maximal length of a simple path for which exactly its last state is a bad state is also an upper bound on the maximal length of counter examples that need to be searched. We call such paths *terminal*.

In the special case $k = 1$ this technique amounts to check that the *good* states are an inductive invariant of the transition relation. Therefore the technique is also known as k-induction [15]. It seems to be much more successful in practice than forward checking, since it can utilize locality of properties, even if it is just implicitly through the SAT solver, while a forward formulation will need to take all state bits into account.

However, simple paths can be exponentially larger than their corresponding diameters, both in forward and backward reasonings. Therefore the question still remains, whether an approach using QBF reasoning would not allow to terminate the search for counter examples much earlier. Also the state of the art in QBF solver technology improved considerably in recent years [11].

To our knowledge, there are no published results on using QBF for backward reasoning yet. Unfortunately our experimental results for backward reasoning provide a strong indication that similar to the forward reasoning results of [10,17] QBF based fixpoint algorithms can not yet really compete with BDD based or other complete model checking algorithms using SAT as discussed for instance in [1]. Not a single instance was solved that could not be solved with k-induction as well.

On the positive side we provide new compact formulations of bounded model checking problems and also show that in certain cases QBF based reasoning can outperform SAT based reasoning. Our benchmarks will be made publicly available. They will help to improve the state of the art of QBF solvers and hopefully lead to

```
model-check$^{\mu}_{\text{forward}}$ $(I,\ T,\ B)$
    $C = false$; $N = I$;
    while $N \not\Rightarrow C$ do
        if $B \wedge N$ satisfiable then
            return "bad state reachable";
        $C = N$;
        $N = C \vee Img(C)$;
    done;
    return "no bad state reachable";
```

Fig. 1. On-the-fly forward model checking algorithm for safety properties.

efficient QBF based model checking algorithms.

Finally, we experimented with functional and relational unrollings of the next state logic. The experiments clearly show that a functional unrolling is much more compact. The generated CNF is much smaller when using syntactic substitution for next state functions instead of conjoining the transition relations. The run times of the SAT solver also decreases considerably.

2 Background

Quantified boolean formulas (QBF) form a propositional logic with quantifiers over boolean variables. The QBF solvers we use only accept QBF in conjunctive normal form (CNF) in prenex form. The standard algorithm [18] for producing CNF for SAT can also be used for QBF after pulling out the quantifiers. The additional variables will be existentially quantified in the innermost scope. In the rest of the paper we do not require prenex CNF.

Our system model is the standard relational model used in symbolic model checking. It is a flat boolean encoded Kripke structure K with initial state constraint $I(s)$, transition relation $T(s, s')$, bad state constraint $B(s)$ and good state constraints $G(s)$ with $G(s) \equiv \neg B(s)$. Evaluations $\sigma \in 2^n$ of state variable vectors s, s', ... act as states. A state variable vector, in the following just short *state variable*, is made up of n individual *state bits*, which are just boolean variables. Individual state bits and equalities over state variables serve as atomic propositions.

A valid *path* of length k in K is an evaluation of state variables $s_0, \ldots s_k$ that satisfies the *path constraint* $T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k)$. An *initialized path constraint* requires in addition that $I(s_0)$ holds, while a *terminal path constraint* requires that $B(s_k)$ holds. A bad state is reachable iff there is a satisfiable path constraint for some k, which at the same time is initial and terminal. In this paper we only consider the problem of checking, whether a bad state is reachable.

3 Fixpoints

The algorithm of Fig. 1 is the standard BFS algorithm for checking simple safety properties with BDDs. The sets C and N as well as the relations I, T, and B are represented symbolically. The algorithm implements a fixpoint computation starting from the initial states, adding the next states N reachable in one step, with $\mathit{Img}(C)(s') \equiv \exists s[T(s,s') \wedge C(s)]$, from the *current* states reached so far C until either a bad state B is found or the loop terminates. The focus of BMC is the former while in this paper we concentrate on checking the loop condition.

The loop condition is invalid initially, and the loop is not even entered, iff $I = \mathit{false}$, or equivalently if $I(s)$ is unsatisfiable. This can be checked by a SAT solver. The validity of the loop condition, after the first iteration can also be checked by a SAT solver, since it is equivalent to the satisfiability of $\exists s, s'[I(s) \wedge T(s,s') \wedge \neg I(s')]$. If this formula is unsatisfiable then I actually turns out to be an inductive invariant of the transition relation.

However, after the second iteration the loop condition is equivalent to the satisfiability of

$$\begin{aligned}\exists s_0, s_1, s_2[\ & I(s_0) \wedge T(s_0,s_1) \wedge T(s_1,s_2) \wedge \\ & \forall t_0, t_1[I(t_0) \wedge T(t_0,t_1) \rightarrow (s_2 \neq t_0 \wedge s_2 \neq t_1)]]\end{aligned}$$

which is a proper QBF formula with one alternation.[3] In general, the loop condition is fulfilled after k iterations iff the following formula is satisfiable:

$$\begin{aligned}\exists s_0, s_1, \ldots, s_k[\ & I(s_0) \wedge T(s_0,s_1) \wedge \cdots \wedge T(s_{k-1},s_k) \wedge \\ & \forall t_0, t_1, \ldots, t_{k-1}[\ I(t_0) \wedge T(t_0,t_1) \wedge \cdots \wedge T(t_{k-2},t_{k-1}) \rightarrow \\ & \qquad (s_k \neq t_0 \wedge \cdots \wedge s_k \neq t_{k-1})]]\end{aligned}$$

Variations of this formulation, were also used in [10,17]. Their practical usage is rather restricted. There was not a single instance in our experiments, where initialized diameter checking, was doable this way, if it involved any alternation of quantifiers. Clearly much stronger QBF solvers are required.

Initial experiments in using the reoccurrence diameter were also unsuccessful. The instances are solvable for small k, but the reoccurrence diameter turns out to be too large for these examples and the SAT instances also become intractable very soon. This is in contrast to the experience with simple path constraints in k-induction. Therefore we suggest to represent the termination check for symbolic backward fixpoint computation as QBF decision problem as well:

$$\begin{aligned}\exists s_0, s_1, \ldots, s_k[\ & T(s_0,s_1) \wedge \cdots \wedge T(s_{k-1},s_k) \wedge B(s_k) \wedge \\ & \forall t_0, t_1, \ldots, t_{k-1}[\ T(t_0,t_1) \wedge \cdots \wedge T(t_{k-2},t_{k-1}) \wedge B(t_{k-1}) \rightarrow \\ & \qquad (s_0 \neq t_0 \wedge \cdots \wedge s_0 \neq t_{k-1})]]\end{aligned}$$

[3] Here we need the common assumption that the transition relation T is total, which we will assume for the rest of the paper.

In our experiments it turns out that in this case two instances for $k = 2$ could be solved, for which also k-induction determined termination easily. Nevertheless, this negative result still shows that even when using QBF, backward computation may be superior to forward computation as it is the case with checking reoccurrence diameters versus k-induction.

For backward fixpoint calculations we actually used a slightly different formulation, as also used in SAT based k-induction [15], where T is replaced by T_G with $T_G(s, s') \equiv G(s) \wedge T(s, s')$. If the formula is unsatisfiable then k is a bound on the maximum length of paths that have to be searched in order find a path to a bad state, which only traverses good states except for the last state. This optimization may reduce the bounds that have to be checked considerably.

4 Non-Copying Iterative Squaring

Following the classical proof of PSPACE hardness of QBF [13,16] we can use *non-copying iterative squaring* to compute symbolically the transitive closure of the transition relation as follows:

$$\begin{aligned} T^{2\cdot i}(s, s') \;\equiv\; \exists\, m\, [\, \forall c\, [\, \exists\, l, r\, [\, (c \rightarrow (\;\, l = s \;\, \wedge r = m)) \;\wedge \\ (\bar{c} \rightarrow (\;\, l = m \wedge r = s')) \;\wedge\; T^i(l, r)]]] \end{aligned}$$

The universal "choice variable" c just instantiates the formal parameters (l, r) of $T^i(l, r)$ with either the actual parameters (s, m) in the positive case or (m, s') in the negative case. This is simply a compact QBF reformulation of *copying iterative squaring*

$$T^{2\cdot i}(s, s') \;\equiv\; \exists\, m\, [\, T^i(s, m) \wedge T^i(m, s')]$$

which doubles the size of the formula with every application, while the non-copying formulation just adds some state variable equalities each time.

A similar formulation was discussed in [12] and has also been used in [2] to perform bounded model checking of very simple counter circuits. In the latter paper it has been observed that current state-of-the-art QBF solvers can barely keep up with SAT based bounded model checking on these examples. However, the QBF formula is linear in the model and logarithmic in the number of steps, which gives an at most quadratic formula in the number of state bits. The worst case only occurs if the sequential depth, e.g. the initialized diameter, really turns out to be 2^n.

5 Simple Path Constraints

In [3,15] the concept of *simple path constraints* was introduced

(1) $$\bigwedge_{0 \leq i < j \leq k} s_i \neq s_j$$

Note that the size of this formula is quadratic in k and each $s_i \neq s_j$ involves the comparison of n state bits. By sorting the s_i symbolically as in [9] an $\mathcal{O}(k \cdot log(k))$

size bound can be obtained. In practice, due to large constants, simpler sorting networks with size $\mathcal{O}(k \cdot (log(k))^2)$ are preferred, such as *bitonic sort* or *odd-even mergesort*. In our experiments we used the latter, since it requires slightly less comparisons than the bitonic sorting network used in [9].

If these constraints are conjoined with path constraints, they allow to obtain a complete model checking procedure. If the path constraints are initialized and the result becomes unsatisfiable, then k is a bound on the reoccurrence diameter [3]. If no counter example up to this length exists, no bad state is reachable. Similarly, if the simple path constraints are conjoined with a terminal path constraint, as in k-induction [15], then the unsatisfiability of the result, again shows that k is an upper bound on the maximal length of counter examples that need to be considered. The formula that is checked in k-induction is the following:

$$(2) \qquad \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \;\wedge\; \bigwedge_{i=0}^{k-1} G(s_i) \;\wedge\; B(s_k) \;\wedge \bigwedge_{0 \le i < j < k} s_i \neq s_j$$

Note, that the last state s_k as a "good state" can never be equal to one of the previous "bad states". Therefore, from Eqn. 1 we can remove comparisons with the last state in k-induction. A similar argument could be used for computing the reoccurrence diameter.

5.1 Compact Simple Path Constraints in QBF

One of our contribution of this paper is a reformulation of the simple path constraints of Eqn. 1 in QBF as follows:

$$(3) \qquad \forall l_0, \ldots, l_k \;[\; \exists s \;[\; |\sum_{i=0}^{k} l_i| = 1 \;\rightarrow\; \bigwedge_{i=0}^{k} (l_i \;\leftrightarrow\; (s = s_i))]$$

The resulting formula needs one alternation of quantifiers and is linear in k as opposed to quadratic complexity of the original formula. Note that the state variables of the corresponding path constraints are free variables of this formula and are quantified existentially in the outermost scope for our applications.

In order to obtain linear complexity the cardinality constraint $|\Sigma_{i=0}^{k} l_i|$, which simply states that exactly one of the l_i is true, has to be encoded with a linear sized circuit. This is easily possible, since for instance the ROBDD for this cardinality constraint for any variable order has linear size in k.

The additional "bits" $l_0, \ldots, l_k$ provide a one-hot encoding of the index of a reference vector, which is saved in s and is enforced to be different from all the other state vectors, e.g. l_i saves s_i as s and forces s to be different from all other s_j with $i \neq j$. A binary encoding of the index of the reference vector is also possible and requires only $\lceil log_2 k \rceil$ additional universal variables.

This example already shows some of the modelling power of QBF, which, we believe, is hardly used in practice yet. But we can go one step further by sharing the transition relation across time frames as in [6]. Our QBF reformulation following

[6] of path constraints is as follows:

$$\forall l_0, \ldots, l_k \;[\; \exists s, s' \;[\; T(s, s') \wedge (|\textstyle\sum_{i=0}^{k} l_i| = 1 \;\rightarrow\; \bigwedge_{i=0}^{k-1}(l_i \;\rightarrow\; (s = s_i \wedge s' = s_{i+1})))]] \tag{4}$$

Putting both together we obtain a compact reformulation of simple path constraints in QBF with transition relation sharing:

$$\forall l_0, \ldots, l_k \;[\; \exists s, s' \;[\; T(s, s') \wedge (|\textstyle\sum_{i=0}^{k} l_i| = 1 \;\rightarrow\; \bigwedge_{i=0}^{k-1}(l_i \;\rightarrow\; (s = s_i \wedge s' = s_{i+1})) \;\wedge\; \bigwedge_{i=0}^{k}(l_i \;\leftrightarrow\; (s = s_i)))]] \tag{5}$$

Initial respectively terminal constraints can be added as needed. In the context of k-induction practical experience shows that adding good state constraints to the current state of the transition relation, as in Eqn. 2, improves performance and particularly decreases the bound k considerably. We can achieve the same effect in the QBF formulation by just adding $G(s)$ to the innermost existential scope, thus, actually sharing G across time frames as well. In the experiments we used the latter version.

6 Transition Functions and Relations

SMV [5] allows two ways to specify the transitions that a system can make. We refer to these as functional and relational part. In the functional part (inside an `ASSIGN` section of the SMV file) the value of a variable in the next state is defined as a boolean function of the variable values in the current state. The relational part is simply a boolean formula where the atomic formulas are current and next state variables and this formula (given in the `TRANS` section of the SMV file) has to hold between any two states. An SMV file can contain both a relational and functional part to describe the system's transition relation.

A functional transition relation allows an optimization in the translation to SAT/QBF when the transition relation (or the simple path constraint) is unrolled. Namely, for a functional state variable, it is possible to substitute its next state function in any state after the initial state. Thereafter, the representation can be simplified by propagating information from the initial state to subsequent states. Consider for instance variables x_0 and x_1 and let the SMV file contain the definitions `init(x0) := 0` and `next(x1) := !x0`. Then it is obviously possible to infer that the value of x_1 after the transition relation is unrolled once is 1 etc. We refer to this optimization as *functional substitution.* Notice that this substitution is not possible if QBFs are used to share the transition relation and the simple state constraint. In our experiments, we compare this optimized translation to a translation where functional substitution is disabled (considering the model be purely relational). Our experimental results show that in some cases the optimization that functionality allows plays an important role.

7 Experiments

We have implemented our approach in a tool called SMV2QBF. It reads flat SMV specifications with simple safety properties as input and translates them to QBFs. The tool has several switches corresponding to different model checking problems. It is possible to perform standard BMC, compute diameter and reoccurrence diameter, compute fixpoint, and do k-induction proofs [8]. For most of these problems, there are two or more encodings, the standard propositional one and one using more compact QBFs.

We present two sets of results, first of problems where it is possible to prove that the safety property holds using k-induction (the induction step eventually becomes unsatisfiable). Second, we have examples where a counterexample is found using standard BMC.

For the experiments we used a cluster of Pentium IV 3.0 GHz PCs with 2GB of main memory running Debian Sarge Linux. The time limit was set to 1000 seconds and the memory limit to 1GB of main memory. The examples that we use are from the TIP tool by Eén and Sörensson [8]. We use QUANTOR (version 2.13) [2] as the QBF solver. QUANTOR uses a SAT solver as a back end and for this purpose we use PICOSAT (version 1.251). We also compare QUANTOR to another state-of-the-art QBF solver, QUBE (version 1.3) [7]. For every instance we tried, QUANTOR performed better.

The results for the examples where the property is proven are shown in Tables 1 and 2. The columns of the tables are as follows. In both tables, the two leftmost columns give the name of the example and the bound needed to prove the property. Thereafter are 6 columns of the form s(x) (Table 1) and t(x) (Table 2), where x is the type of encoding, s(x) stands for size and t(x) for time. We use the following encodings for k-induction step:

(i) i is the standard fully propositional encoding (Eqn. 2),

(ii) ir is i without functional substitution (see Sect. 6),

(iii) is implements simple state constraints with sorting networks,

(iv) isr is is without functional substitution,

(v) l is an encoding where the transition relation is unrolled but the simple path constraints are encoded as given in Eqn. 3,

(vi) lr is l without functional substitution,

(vii) L is an encoding using Eqn. 5 using a one-hot index encoding, and

(viii) B is a modification of Eqn. 5 with binary index encoding.

A column of the form s(x) gives the size in kilobytes of the (SAT/QBF) formula for encoding x and the bound given in column k. The running time given in column t(x) is the time required to solve the *single* instance of encoding x corresponding to the depth given in the column k. If the entry is of the form N/A then the memory limit was exceeded (we experienced no time outs).

name	k	s(i)	s(ir)	s(is)	s(isr)	s(l)	s(lr)	s(L)	s(B)
cmu.periodic.N	96	41372	41416	27996	28096	9832	9844	2100	2112
eijk.S208.S	258	146728	170828	49772	72808	3232	26864	3656	3696
eijk.S208c.S	258	148840	186168	51332	82716	3212	33792	3884	3924
eijk.S208o.S	258	129788	164740	44240	77480	3048	37516	2920	2956
eijk.S298.S	58	13868	19752	10740	16584	1136	6812	1668	1672
eijk.S510.S	10	656	2504	1268	3068	372	2404	632	636
eijk.S820.S	11	844	3468	1300	3904	584	3500	664	664
eijk.S832.S	11	900	3592	1400	4072	628	3704	700	700
eijk.S953.S	7	448	2612	748	2912	360	2748	776	776
ken.oop1.C	29	3204	4204	3536	4504	1116	2184	608	608
nusmv.guid*1.C	10	1360	2564	2192	3320	1112	2224	752	752
nusmv.guid*7.C	27	8208	11996	9520	13316	2712	6172	1728	1732
nusmv.tcas2.B	6	1176	3936	1820	4584	1016	4364	1284	1288
nusmv.tcas3.B	5	892	2964	1420	3704	836	3640	1172	1172
texas.par*2.E	2	36	480	44	488	44	548	356	356
vis.prodc*12.E	29	10612	65488	11352	66272	6008	68328	3320	3320
vis.prodc*13.E	8	1556	16112	1884	16488	1472	18792	2444	2444
vis.prodc*14.E	16	4112	33700	4776	34440	3164	37312	2776	2776
vis.prodc*15.E	23	7260	49808	8496	51024	4664	53496	3068	3068
vis.prodc*16.E	5	800	9752	1004	9980	824	11740	2320	2320
vis.prodc*17.E	27	9444	60076	10392	61092	5528	59872	3236	3236
vis.prodc*18.E	13	3036	26284	3708	27044	2500	28616	2652	2652
vis.prodc*19.E	22	6772	47484	8012	48712	4468	51204	3028	3028
vis.prodc*24.E	37	15832	87760	16888	88900	7924	89092	3652	3656

Table 1
k-induction sizes

Tables 3 and 4 follow the same conventions as Tables 1 and 2, however, this time k is the smallest depth needed to find a counterexample. The encodings are as follows:

(i) b, the standard propositional BMC encoding,

(ii) br is b without functional substitution,

(iii) C, a compact BMC encoding with a single copy of the transition relation (see Eqn. 4), and

(iv) S, a BMC encoding with noncopying iterative squaring (see Sect. 4).

Tables 1–4 seem to warrant the following conclusions. Applying QBF representations yields in many cases smaller formulas and the difference seems to grow with larger bounds. This is especially so when the model is fully relational. This conclusion is rather obvious, though, since the QBF grows linearly and in the propositional case a new copy of the transition relation is needed when the bound is incremented.

A perhaps more interesting observation is that the running times of the optimized version of the propositional encodings (columns t(i) and t(b)) are always lower than the encodings using more compact formulas. We identify two reasons for this. First, propositional encoding allows one to perform optimizations (prepro-

name	k	t(i)	t(ir)	t(is)	t(isr)	t(l)	t(lr)	t(L)	t(B)
cmu.periodic.N	96	144.7	144.6	178.4	173.3	162.5	162.3	345.1	153.5
eijk.S208.S	258	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
eijk.S208c.S	258	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
eijk.S208o.S	258	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
eijk.S298.S	58	66.0	61.5	N/A	N/A	N/A	207.7	195.1	132.2
eijk.S510.S	10	1.9	2.4	163.7	174.8	N/A	5.4	5.7	5.9
eijk.S820.S	11	2.2	3.4	93.8	82.1	N/A	5.3	3.6	3.4
eijk.S832.S	11	2.3	3.5	95.5	84.9	N/A	6.0	3.9	3.6
eijk.S953.S	7	0.8	1.7	8.6	8.3	38.4	2.8	3.1	2.8
ken.oop1.C	29	23.0	18.3	N/A	N/A	30.4	30.9	50.3	37.8
nusmv.guid*1.C	10	1.4	2.0	1.5	2.1	6.3	6.2	7.6	6.2
nusmv.guid*7.C	27	61.7	60.2	173.7	85.8	87.5	92.6	116.5	116.7
nusmv.tcas2.B	6	1.2	2.5	1.3	2.6	3.9	4.9	11.1	6.4
nusmv.tcas3.B	5	0.5	1.4	1.0	2.0	3.2	3.4	6.5	4.9
texas.par*2.E	2	0.0	0.2	0.0	0.2	0.1	0.2	0.4	0.2
vis.prodc*12.E	29	30.3	197.9	25.0	173.1	N/A	245.8	74.3	57.0
vis.prodc*13.E	8	1.4	13.3	1.6	12.6	5.0	16.8	8.6	8.1
vis.prodc*14.E	16	5.6	46.6	4.6	44.4	23.7	64.6	20.2	17.6
vis.prodc*15.E	23	15.3	113.5	17.3	100.5	85.0	146.2	43.8	34.4
vis.prodc*16.E	5	0.8	6.2	0.9	6.2	2.1	8.6	7.4	6.9
vis.prodc*17.E	27	32.1	163.7	25.2	145.6	N/A	200.2	64.6	52.1
vis.prodc*18.E	13	3.6	28.7	3.7	29.6	13.9	37.9	14.5	12.7
vis.prodc*19.E	22	12.6	97.6	12.3	94.4	70.6	130.1	36.7	32.0
vis.prodc*24.E	37	59.8	N/A	49.4	N/A	N/A	N/A	125.0	103.9

Table 2
k-induction running times

name	k	s(b)	s(br)	s(C)	s(S)
nusmv.tcas1.B	10	960	5580	4512	1524
nusmv.tcas4.B	14	1604	9256	6496	1516
nusmv.tcas5.B	23	2688	13104	9796	1668
nusmv.tcas6.B	16	2816	14900	10188	1668
texas.parsesys1.E	9	140	2392	2140	568
texas.parsesys3.E	8	100	2088	1812	516
texas.twoproc2.E	15	48	13832	9676	1636
texas.twoproc4.E	19	224	19004	12408	1676
vis.eisenberg.E	19	632	19644	12172	1580

Table 3
BMC sizes

cessing steps), like functional substitution (see Sect. 6) but also bounded cone of influence [4] in an efficient manner. [4] Indeed, it should be noted that for some test cases (like the examples vis.prodcell.*), when the SMV model is made fully relational and thus no functional substitutions are possible, QBF encodings sharing the transition relation (columns t(L) and t(B)) perform better than the SAT encoding (column t(ir)).

Second, the research community has invested much more effort to implement efficient SAT solvers than is the case for QBFs. We expect more efficient QBF

[4] Notice that we always reduce the model by cone of influence reduction in every encoding, particularly for the transition relation and comparing state variables. In addition, we only compare state variables that occur in both current and next states [8].

name	k	t(b)	t(br)	t(C)	t(S)
nusmv.tcas1.B	10	0.9	42.3	364.6	56.7
nusmv.tcas4.B	14	1.6	46.8	558.0	56.7
nusmv.tcas5.B	23	3.3	51.4	N/A	81.9
nusmv.tcas6.B	16	3.2	46.9	N/A	76.1
texas.parsesys1.E	9	0.1	10.2	86.4	10.8
texas.parsesys3.E	8	0.1	9.1	69.0	8.8
texas.twoproc2.E	15	0.0	133.4	N/A	141.8
texas.twoproc4.E	19	0.3	147.4	N/A	213.3
vis.eisenberg.E	19	1.1	80.9	N/A	117.5

Table 4
BMC running times

solvers in the future.

8 Conclusion

This paper on one hand again provides negative results on using QBF for unbounded model checking and less negative for bounded model checking. On the other hand we were able to show that in practice QBF formulations can be much more compact than SAT instances and sometimes solved faster for relational encodings. Our results clearly show that much more research in QBF is needed to be able to use QBF as alternative to SAT based model checking, even in the bounded case.

The tool SMV2QBF and the benchmarks in DIMACS format are available at http://fmv.jku.at/smv2qbf. We are currently working on producing structural benchmarks as well, in the form of *and-inverter-graphs* (AIGs).

References

[1] N. Amla, X. Du, A. Kuehlmann, R. P. Kurshan, and K. L. McMillan. An analysis of SAT-based model checking techniques in an industrial environment. In *Proc. CHARME'05*, volume 3725 of *LNCS*. Springer.

[2] A. Biere. Resolve and expand. In *Proc. SAT'04*, volume 3542 of *LNCS*. Springer.

[3] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. TACAS'99*, volume 1579 of *LNCS*. Springer.

[4] A. Biere, E. Clarke, R. Raimi, and Y. Zhu. Verifying safety properties of a PowerPC microprocessor using symbolic model checking without BDDs. In *Proc. CAV'99*, volume 1633 of *LNCS*.

[5] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proc. CAV'02*, volume 2404 of *LNCS*.

[6] N. Dershowitz, Z. Hanna, and J. Katz. Bounded model checking with QBF. In *Proc. SAT'05*, volume 3569 of *LNCS*. Springer.

[7] A. Tacchella E. Giunchiglia, M. Narizzano. System description: QuBE A system for deciding quantified boolean formulas satisfiability. In *Proc IJCAR'01*, volume 2083 of *LNCS*.

[8] N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. In *Proc. BMC'03*, volume 89 of *ENTCS*. Elsevier.

[9] D. Kröning and O. Strichman. Efficient computation of recurrence diameters. In *Proc. VMCAI'03*, volume 2575 of *LNCS*. Springer.

[10] M. Mneimneh and K. Sakallah. Computing vertex eccentricity in exponentially large graphs: QBF formulation and solution. In *Proc. SAT'03*, volume 2919 of *LNCS*. Springer.

[11] M. Narizzano, L. Pulina, and A. Tacchella. Report of the third QBF solvers evaluation. *Journal on Satisfiability, Boolean Modeling and Computation*, 2, 2006.

[12] 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.

[13] W. J. Savitch. Relation between nondeterministic and deterministic tape complexity. *Journal of Computer and System Sciences*, 4, 1970.

[14] Viktor Schuppan and Armin Biere. Efficient reduction of finite state model checking to reachability analysis. *Software Tools for Technology Transfer (STTT)*, 5(1-2):185–204, March 2004.

[15] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In *Proc. FMCAD'00*, volume 1954 of *LNCS*. Springer.

[16] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *5th Annual ACM Symposium on the Theory of Computing*, 1973.

[17] Daijue Tang, Yinlei Yu, Darsh Ranjan, and Sharad Malik. Analysis of search based algorithms for satisfiability of quantified boolean formulas arising from circuit state space diameter problems. In *Proc. SAT'04*, volume 3542 of *LNCS*. Springer.

[18] G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part II*, volume 8 of *Seminars in Mathematics*. V.A. Steklov Mathematical Institute, 1968.