Extended Resolution Proofs for Symbolic SAT Solving with Quantification

Toni Jussila, Carsten Sinz, and Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University Linz, Austria
{toni.jussila, carsten.sinz, armin.biere}@jku.at

Abstract. Symbolic SAT solving is an approach where the clauses of a CNF formula are represented using BDDs. These BDDs are then conjoined, and finally checking satisfiability is reduced to the question of whether the final BDD is identical to false. We present a method combining symbolic SAT solving with BDD quantification (variable elimination) and generation of extended resolution proofs. Proofs are fundamental to many applications, and our results allow the use of BDDs instead of—or in combination with—established proof generation techniques like clause learning. We have implemented a symbolic SAT solver with variable elimination that produces extended resolution proofs. We present details of our implementation, called EBDDRES, which is an extension of the system presented in [1], and also report on experimental results.

1 Introduction

Propositional logic decision procedures [2–6] lie at the heart of many applications in hard- and software verification, artificial intelligence and automatic theorem proving [7–11], and have been used to successfully solve problems of considerable size. In many practical applications it is not sufficient to obtain a yes/no answer from the decision procedure, however. Either a model, representing a sample solution, or a justification why the formula possesses none is required. In the context of model checking proofs are used, e.g., for abstraction refinement [11] or approximative image computations through interpolants [12]. Proofs are also important for certification by proof checking [13], in declarative modeling [9], or product configuration [10].

Using BDDs for SAT is an active research area [14–19]. It turns out that BDD and search based techniques are complementary [20–22]. There are instances for which one works better than the other. Therefore, combinations have been proposed [15, 16, 19] to obtain the benefits of both, usually in the form of using BDDs for preprocessing. However, in all these approaches where BDDs have been used, proof generation has not been possible so far.

In [1], we presented a method for symbolic SAT solving that produces extended resolution proofs. However, in that paper the only BDD operation considered is conjunction. Here, we address the problem of existential quantification left open in [1]. In particular, we demonstrate how BDD quantification can be combined with the construction of extended resolution proofs for unsatisfiable instances. Using quantification allows to build algorithms that have an exponential run-time only in the width of the elimination order used [17, 21]. It can therefore lead to much faster results on appropriate instances and hence produce shorter proofs, which is also confirmed by our experiments. For instance, we can now generate proofs for some of the Urquhart problems [23].

2 Theoretical Background

We assume that we are given a formula in CNF that we want to refute by an extended resolution proof. In what follows, we largely use an abbreviated notation for clauses, where we write $(l_1 \ldots l_k)$ for the clause $l_1 \vee \cdots \vee l_k$.

We assume that the reader is familiar with the resolution calculus [24]. Extended resolution [25] enhances the ordinary resolution calculus by an *extension rule*, which allows introduction of definitions (in the form of additional clauses) and new (defined) variables into the proof. Additional clauses must stem out of the CNF conversion of definitions of the form $x \leftrightarrow F$, where F is an arbitrary formula and x is a new variable, i.e. a variable neither occurring in the formula we want to refute nor in previous definitions nor in F. In this paper—besides introducing variables for the Boolean constants true and false—we only define new variables for if-then-else (*ITE*) constructs. *ITE*(x, a, b) is the same as $x\ ?\ a : b$ (for variables x, a, b), which is an abbreviation for $(x \rightarrow a) \wedge (\neg x \rightarrow b)$. So introducing a new variable w as an abbreviation for $ITE(x, a, b)$ results in the additional clauses $(\bar{w}\bar{x}a)$, $(\bar{w}xb)$, $(w\bar{x}\bar{a})$ and $(wx\bar{b})$, which may then be used in subsequent resolution steps. Extended resolution is among the strongest proof systems available and equivalent in strength to extended Frege systems [26].

Binary Decision Diagrams (BDDs) [27] are used to compactly represent Boolean functions as directed acyclic graphs. In their most common form as reduced ordered BDDs (that we also adhere to in this paper) they offer the advantage that each Boolean function is uniquely represented by a BDD, and thus all semantically equivalent formulae share the same BDD. BDDs are based on the Shannon expansion $f = ITE(x, f_1, f_0)$, decomposing f into its *co-factors* f_0 and f_1 (w.r.t variable x). The co-factor f_0 (resp. f_1) is obtained by setting variable x to false (resp. true) in formula f and subsequent simplification.

In [1], we presented a symbolic SAT solver that conjoins all the BDDs representing the clauses. This approach has the potential hurdle that the intermediate BDDs may grow too large. If memory consumption is not a problem, however, the BDD approach can be orders of magnitude faster than DPLL-style implementations [17, 18, 20]. Using existential quantification can speed up satisfiability checking even more and, moreover, improve memory consumption considerably by eliminating variables from the formula and thus produce smaller BDDs.

If the formula is a conjunction, rules of quantified logic allow existential quantification of variable x to be restricted to those conjuncts where x actually appears, formally:

$$\exists x(f(x, Y) \wedge g(Z)) = (\exists x f(x, Y)) \wedge g(Z)$$

where Y and Z are sets of variables not containing x. This suggests the following satisfiability algorithm [17]. First, choose a total order $\pi = (x_1, \ldots, x_n)$ of the variables X of formula F. Then, build for each variable x_i a *bucket*. The bucket B_i for x_i initially contains the BDD representations of all the clauses where x_i is the first variable according to π. Start from bucket B_1 and build the conjunction BDD b of all its elements. Then, compute $\exists x_1 b$ and put the resulting BDD to the bucket of its first variable according to π. Then, the computation proceeds to B_2 and continues until all buckets have been processed. If for any bucket, the conjunction of its elements is the constant false, we know that F is unsatisfiable. If the instance is satisfiable we get the true BDD after processing all the buckets.

3 Proof Construction

As above, we assume that we are given a formula F in CNF and that F contains the variables $\{x_1, \ldots, x_n\}$. Furthermore, we assume a given variable ordering π and that the BDD representation of clauses are initially divided into buckets $B_1, \ldots, B_n$ according to π and that variables in the BDDs are ordered according to π (the first variable of π is the root etc.). The details of how clauses are converted to BDDs are given in [1].

Our computation builds intermediate BDDs for the buckets one by one in the order mandated by π. Assume that we process a bucket that contains the BDDs $b_1, \ldots, b_m$. We construct intermediate BDDs h_i corresponding to partial conjunctions of $b_1 \wedge \cdots \wedge b_i$ until, by computing h_m, we have computed a BDD for the entire bucket. Finally, we compute a BDD $\exists h_m$ corresponding to h_m where its root variable has been existentially quantified, and add the BDD $\exists h_m$ to the (so far unprocessed) bucket of its root variable. Assuming that the children of h_m are called h_{m0} and h_{m1}, respectively, these intermediate BDDs can be computed recursively by the equations:

$$h_2 \leftrightarrow b_1 \wedge b_2, \quad h_i \leftrightarrow h_{i-1} \wedge b_i \quad \text{for } 3 \leq i \leq m \quad \text{and} \quad \exists h_m \leftrightarrow h_{m0} \vee h_{m1}$$

If it turns out that h_m is the false BDD, F is unsatisfiable construction of the proof can start. For this construction, we introduce variables (using the extension rule) for each BDD node that is generated during the BDD computation, i.e. for all b_i, h_i, and $\exists h_m$. Let f be such an internal node with the children f_0 and f_1 (leaf nodes are handled according to [1]). Then we introduce a variable (also called f) based on Shannon expansion as follows:

$$f \;\leftrightarrow\; (x \;?\; f_1 : f_0) \qquad\qquad (\bar{f}\bar{x}f_1)(\bar{f}xf_0)(f\bar{x}\bar{f}_1)(fx\bar{f}_0)$$

On the right, we have also given the clausal representation of the definition. In order to prove F, we have to construct proofs of the following formulas for all buckets:

$$\begin{aligned}
&F \vdash b_i && \text{for all } 1 \leq i \leq m && \text{(ER-1)}\\
&F \vdash b_1 \wedge b_2 \rightarrow h_2 && && \text{(ER-2a)}\\
&F \vdash h_{i-1} \wedge b_i \rightarrow h_i && \text{for all } 3 \leq i \leq m && \text{(ER-2b)}\\
&F \vdash h_{m0} \vee h_{m1} \rightarrow \exists h_m && && \text{(ER-3a)}\\
&F \vdash h_m \rightarrow \exists h_m && && \text{(ER-3b)}\\
&F \vdash \exists h_m && && \text{(ER-4)}
\end{aligned}$$

Here, the elements b_i can either be (initially present) clauses or results of an existential quantification. For clauses, the proof is straightforward (see [1]). For non-clauses, the proof is ER-4 (shown below). The proofs of ER-2a, and ER-2b are also given in [1] and we now concentrate on proving ER-3a, ER-3b, and ER-4. For the proof of ER-3a, we use the fact that $\exists h_m$ is the disjunction of the children (we call them h_{m0} and h_{m1}) of h_m. We first prove that $h_{m0} \vee h_{m1} \rightarrow \exists h_m$, in clausal form $(\bar{h}_{m0}\exists h_m)(\bar{h}_{m1}\exists h_m)$. For representational purposes, assume $h_{m0} = f$, $h_{m1} = g$, and $\exists h_m = h$, and that the root variable of f, g and h is x. We know that:

$$\begin{aligned}
f &\;\leftrightarrow\; (x \;?\; f_1 : f_0) && (\bar{f}\bar{x}f_1)(\bar{f}xf_0)(f\bar{x}\bar{f}_1)(fx\bar{f}_0)\\
g &\;\leftrightarrow\; (x \;?\; g_1 : g_0) && (\bar{g}\bar{x}g_1)(\bar{g}xg_0)(g\bar{x}\bar{g}_1)(gx\bar{g}_0)\\
h &\;\leftrightarrow\; (x \;?\; h_1 : h_0) && (\bar{h}\bar{x}h_1)(\bar{h}xh_0)(h\bar{x}\bar{h}_1)(hx\bar{h}_0)\;.
\end{aligned}$$

We now recursively construct an ER proof for $f \vee g \to h$, where in the recursive step we assume that proofs for both $f_0 \vee g_0 \to h_0$ and $f_1 \vee g_1 \to h_1$ are already given. We prove $f \vee g \to h$ by generating separate proofs for $(\bar{f}h)$ and $(\bar{g}h)$. The proof for $(\bar{f}h)$ is as follows.

$$\dfrac{\dfrac{(hx\bar{h}_0) \qquad \dfrac{(\bar{f}xf_0) \quad \dfrac{\vdots}{(\bar{f}_0 h_0)}}{(\bar{f}xh_0)}}{(\bar{f}xh)} \qquad \dfrac{\dfrac{\dfrac{\vdots}{(\bar{f}_1 h_1)} \quad (\bar{f}\bar{x}f_1)}{(\bar{f}\bar{x}h_1)} \qquad (h\bar{x}\bar{h}_1)}{(\bar{f}\bar{x}h)}}{(\bar{f}h)}$$

The recursive process stops when we arrive at the leaf nodes resp. the base case of the recursive *BDD-or* algorithm. The proof for $(\bar{g}h)$ is the same, except that f, f_0, and f_1 are replaced with g, g_0, and g_1, respectively.

The case ER-3b, in clausal form $(\bar{h}_m \exists h_m)$, is not recursive but consists of just three simple steps. The proof uses the results of ER-3a, i.e. $(\bar{h}_{m0} \exists h_m)$ and $(\bar{h}_{m1} \exists h_m)$. The root variables of h_m and $\exists h_m$ are different. To illustrate this we use w instead of x.

$$\dfrac{\dfrac{(\bar{h}_m w h_{m0}) \quad (\bar{h}_{m0} \exists h_m)}{(\bar{h}_m w \exists h_m)} \qquad \dfrac{(\bar{h}_{m1} \exists h_m) \quad (\bar{h}_m \bar{w} h_{m1})}{(\bar{h}_m \bar{w} \exists h_m)}}{(\bar{h}_m \exists h_m)}$$

The proof of ER-4 is just a combination of parts one to three. First, having unit clauses b_1 and b_2, we resolve h_2 (using ER-2a), then all the h_i up to h_m (using ER-2b) and finally $\exists h_m$ (using ER-3b). The so-produced proofs may contain tautological clauses. As stated in [1] for the case of conjunction, careful analysis is needed in order to remove them, but it is clearly possible, also in case of existential quantification (disjunction). The full details will be given in an extended version.

4 Implementation and Experimental Result

We have implemented our approach in the SAT solver EBDDRES. It takes as input a CNF formula in DIMACS format and computes the bucket elimination algorithm. The result is either the false BDD or the true BDD. In the latter case, a satisfying assignment is created by traversing the intermediate BDDs right before existential quantification (called h_m above) from the last eliminated variable to the first. For the last eliminated variable, a truth value is chosen based on which branch of the BDD leads to the sink true. For all the previous BDDs, the value for the root variable is chosen based on seeking from its children a path to the sink true. Notice that for all the variables below the root, the truth value is already fixed. Therefore, at maximum two paths have to be traversed for each root h_m. The length of the traversed paths grow from one to the number of variables in the worst case. Thus, the algorithm to find a satisfiable valuation is quadratic in the number of variables. In practise with our test cases, this has not been a problem. Finally, for unsatisfiable cases a proof trace (deduction of the empty clause) can be generated.

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 main memory. No limit was imposed on the generated traces. The experimental results are presented in Table 1.

Table 1. Comparison of Trace generation with MINISAT and with EBDDRES.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	MINISAT			EBDDRES							EBDDRES, quantification						
	solve resources		trace size	solve resources		trace				bdd nodes	solve resources		trace				bdd nodes
	sec	MB	MB	sec	MB	gen sec	ASCII MB	bin MB	chk sec	$\times 10^3$	sec	MB	gen sec	ASCII MB	bin MB	chk sec	$\times 10^3$
ph7	0	0	0	0	0	0	1	0	0	3	0	5	0	12	4	1	60
ph8	0	4	1	0	0	0	3	1	0	15	1	14	1	49	15	4	236
ph9	6	4	11	0	0	0	3	1	0	8	6	52	4	186	59	14	864
ph10	44	4	63	1	17	1	30	10	2	136	20	214	16	683	*	*	2974
ph11	884	6	929	1	13	1	21	8	2	35	-	*	-	-	-	-	-
ph12	*	-	-	2	22	1	33	12	3	31	-	*	-	-	-	-	-
ph13	*	-	-	10	126	7	260	92	20	850	-	*	-	-	-	-	-
ph14	*	-	-	9	111	7	204	74	18	166	-	*	-	-	-	-	-
mutcb8	0	0	0	0	0	0	2	1	0	10	0	0	0	3	1	0	16
mutcb9	0	4	0	0	5	0	5	2	0	27	0	4	0	6	2	0	35
mutcb10	0	4	1	0	8	0	11	4	1	58	0	5	0	11	4	1	59
mutcb11	1	4	4	1	17	1	31	10	2	153	1	8	1	23	7	2	123
mutcb12	8	4	22	2	32	2	69	22	5	320	1	13	1	38	12	3	198
mutcb13	112	5	244	7	126	5	181	61	13	817	2	24	2	70	22	5	347
mutcb14	488	8	972	14	250	10	393	132	27	1694	4	37	3	127	40	8	621
mutcb15	*	-	-	36	498	26	1009	*	*	4191	6	52	5	211	67	14	1012
mutcb16	*	-	-	-	*	-	-	-	-	-	12	104	9	391	126	26	1821
urq35	95	4	218	2	22	1	37	13	3	24	0	0	0	1	0	0	5
urq45	*	-	-	-	*	-	-	-	-	-	0	0	0	1	0	0	10
urq55	*	-	-	-	*	-	-	-	-	-	0	0	0	2	1	0	15
urq65	*	-	-	-	*	-	-	-	-	-	0	4	0	6	2	0	34
urq75	*	-	-	-	*	-	-	-	-	-	0	4	0	7	2	0	39
urq85	*	-	-	-	*	-	-	-	-	-	0	5	0	10	3	1	59
fpga108	0	2		6	47	4	135	47	11	186	8	92	6	239	77	18	1088
fpga109	0	0		3	44	2	70	24	6	83	10	114	8	323	105	9	1434
fpga1211	0	0		53	874	37	1214	*	*	1312	-	*	-	-	-	-	-
add16	0	0	0	0	4	0	6	2	0	30	0	3	0	4	2	0	26
add32	0	0	0	1	9	1	24	8	2	122	1	7	0	19	6	1	106
add64	0	0	0	12	146	9	338	112	23	1393	12	95	9	393	127	26	1839
add128	0	4	0	-	*	-	-	-	-	-	-	*	-	-	-	-	-

The first column lists the name of the instance (see [1] for descriptions of the instances). Columns 2-4 contain data for MINISAT, first the time taken to solve the instance including the time to produce the trace, then the memory used, and in column 4 the size of the generated trace. The data for EBDDRES takes up the rest of the table, columns 5-11 for the approach only conjoining BDDs [1] and 12-18 for variable elimination. Column 5 (12) shows the time taken to solve the instance with EBDDRES including the time to generate and dump the trace. The latter is shown separately in column 7 (14). The memory used by EBDDRES, column 6 (13), is linearly related to the number of BDD nodes shown in column 11 (18). Column 8 (15) shows the size of the trace files in ASCII format. Column 9 (16) shows the size in a binary format comparable to that used by MINISAT (column 4). Finally, column 10 (17) shows the time needed to check that the trace is correct. The * denotes either *time out* ($>$ 1000 seconds) or *out of memory* ($>$ 1GB of main memory). The table shows that quantification performs worse than conjoining on pigeonhole formulas (ph*). We assume that this could be improved if we used separate variable orderings for BDDs and elimination. On the other hand, quantification is faster on the mutilated checker board instances (mutcb*) and Urquhart formulas (urq*).

5 Conclusions

Resolution proofs are used in many practical applications. Our results enable the use of BDDs for these purposes instead—or in combination with—already established methods based on DPLL with clause learning. This paper extends work in [1] by presenting a practical method to obtain extended resolution proofs for symbolic SAT solving with existential quantification. Our experiments confirm that on appropriate instances we are able to outperform both a fast search based approach as well as our symbolic approach only conjoining BDDs.

References

1. C. Sinz and A. Biere. Extended resolution proofs for conjoining BDDs. In *Proc. CSR'06*, 2006.
2. M. Davis and H. Putnam. A computing procedure for quantification theory. *JACM*, 7, 1960.
3. J. P. Marques-Silva and K. A. Sakallah. GRASP — a new search algorithm for satisfiability. In *Proc. ICCAD'96*.
4. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. DAC'01*.
5. E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Proc. DATE'02*.
6. N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT'03*.
7. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. TACAS'99*.
8. M. Velev and R. Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors. *J. Symb. Comput.*, 35(2), 2003.
9. I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan, and M. Taghdiri. Debugging overconstrained declarative models using unsatisfiable cores. In *Proc. ASE'03*.
10. C. Sinz, A. Kaiser, and W. Küchlin. Formal methods for the validation of automotive product configuration data. *AI EDAM*, 17(1), 2003.
11. K. McMillan and N. Amla. Automatic abstraction without counterexamples. In *Proc. TACAS'03*.
12. K. McMillan. Interpolation and SAT-based model checking. In *Proc. CAV'03*, volume 2725 of *LNCS*.
13. L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proc. DATE'03*.
14. D. Motter and I. Markov. A compressed breath-first search for satisfiability. In *ALENEX'02*.
15. J. Franco, M. Kouril, J. Schlipf, J. Ward, S. Weaver, M. Dransfield, and W. Fleet. SBSAT: a state–based, BDD–based satisfiability solver. In *Proc. SAT'03*.
16. R. Damiano and J. Kukula. Checking satisfiability of a conjunction of BDDs. In *DAC'03*.
17. J. Huang and A. Darwiche. Toward good elimination orders for symbolic SAT solving. In *Proc. ICTAI'04*.
18. G. Pan and M. Vardi. Search vs. symbolic techniques in satisfiability solving. In *SAT'04*.
19. H.-S. Jin, M. Awedh, and F. Somenzi. CirCUs: A hybrid satisfiability solver. In *Proc. SAT'04*.
20. T. E. Uribe and M. E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In *Proc. Intl. Conf. on Constr. in Comp. Logics*, volume 845 of *LNCS*, 1994.
21. I. Rish and R. Dechter. Resolution versus search: Two strategies for SAT. *J. Automated Reasoning*, 24(1–2), 2000.
22. J. F. Groote and H. Zantema. Resolution and binary decision diagrams cannot simulate each other polynomially. *Discrete Applied Mathematics*, 130(2), 2003.
23. A. Urquhart. Hard examples for resolution. *J. ACM*, 34(1), 1987.
24. J. A. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12, 1965.
25. G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, 1970.
26. A. Urquhart. The complexity of propositional proofs. *Bulletin of the EATCS*, 64, 1998.
27. R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8), 1986.