Local Two-Level And-Inverter Graph Minimization without Blowup

Robert Brummayer and Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University Linz, Austria
{robert.brummayer, armin.biere}@jku.at

Abstract. And-Inverter Graphs (AIGs) are an efficient and scalable representation for boolean formulas and circuits. We present a maximal set of rules for local two-level optimization of AIGs. This set consists of rules which can be applied before node creation greedily without affecting structural sharing negatively. We implemented these techniques in the AIG library of our tool SMV2QBF and report on experimental results in the context of SAT based model checking.

1 Introduction

Efficient and scalable circuit representations play an important role in synthesis and formal verification of circuits. A common approach is to use Directed Acyclic Graph (DAG) representations which allow to share subcomponents. Examples of such representations are Boolean Expression Diagrams (BEDs) [2], Reduced Boolean Circuits (RBCs) [1] and AIGs [10].

Several algorithms have been developed to optimize such representations. For example [12] presents a DAG-aware AIG rewriting technique for preprocessing combinational logic before technology mapping.

Optimization algorithms can be classified as global or local. Global algorithms operate on the whole DAG, by for instance taking reference counts into account. The goal is to minimize the overall (global) node count by rewriting. However, local algorithms operate only on a small portion of the DAG. The typical example are algorithms that work on a two-level window during node generation, i.e. when a new node has to be generated, the context to which a rewrite rule is applicable consists of the new node, its children and its grand-children. This is the original approach taken in minimizing BEDs, RBCs, and also AIGs [1,2,10].

Local optimization algorithms have to be used with caution because they can affect structural sharing negatively. Figure 1 shows two examples where a local optimization can have negative side effects. In this figure, circles denote AND gates and dots negations.

Consider the rule of distributivity. The unoptimized AIG on the left represents the formula $(a \vee b) \wedge (b \vee d)$ while the right AIG represents $(a \wedge d) \vee b$. From a local point of view, the right AIG is more compact than the left AIG. Unfortunately, this may not be the case from a global point of view. The left

AIG introduces one new node which represents the top level conjunction. However, the AIG on the right side introduces *two* new nodes ignoring the fact that $(a \vee b)$ and $(b \vee d)$ are still referenced. To summarize the example, two nodes are generated while only one is saved. A similar effect is obtained when greedily applying the substitution rule on the right of Fig. 1, which therefore is considered harmful.

Therefore local optimization is not robust and may produce larger circuits. The experiments in [3] show that applying local optimization greedily often even increases the size of the minimized circuit considerably. In one example the size of a 80k node circuit is increased by 50k nodes using all possible local optimization rules greedily.

We present a maximal set of local optimization rules which are guaranteed not to affect structural sharing negatively. These rules operate on a two-level window of the AIG and are a subset of all equivalence preserving two-level rewrite rules.

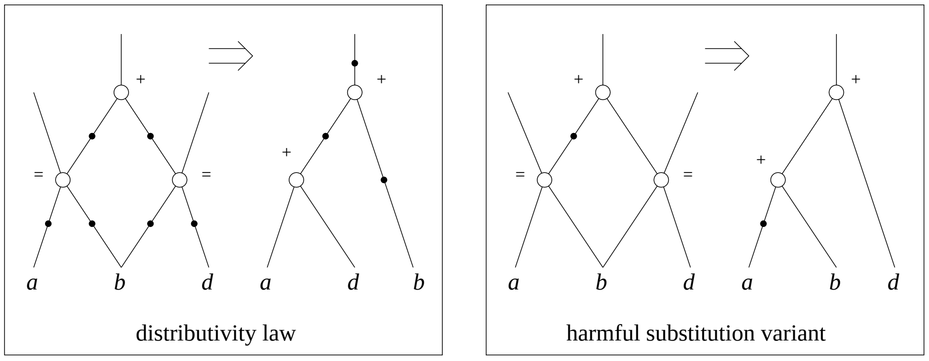

Fig. 1. Optimization rules affecting structural sharing negatively.

2 Background

An And-Inverter Graph (AIG) is a Directed Acyclic Graph (DAG) consisting of constants, primary inputs, two-input gates and inverters. Primary inputs and constants are represented by terminal nodes, AND gates by nodes with two inputs and one output. Inversion is indicated by a complemented edge. Multiple AIGs can be used in parallel to model bit vectors. Unlike Reduced Ordered Binary Decision Diagrams (ROBDDs) [5] AIGs are non-canonical. Thus in general there is no unique representation of a boolean formula as AIG.

AIGs can be used to represent arbitrary boolean formulas and circuits, and are implemented in several logic synthesis and verification systems [11,12]. It is possible to represent every basic logic operation by AND gates and inverters and therefore by AIGs. Table 1 shows how such a representation of the most important basic logic operations could look like.

A hash table is used to remove redundant components during construction by structure sharing. Automatic structure sharing and the simplicity of AIGs make them a compact, simple, easy to use, and scalable representation.

Name	Function	Representation by two-input AND and inversion
Inversion	$\neg x$	$\neg x$
Conjunction	$x \wedge y$	$x \wedge y$
Disjunction	$x \vee y$	$\neg(\neg x \wedge \neg y)$
Implication	$x \rightarrow y$	$\neg(x \wedge \neg y)$
Equivalence	$x \leftrightarrow y$	$\neg(x \wedge \neg y) \wedge \neg(\neg x \wedge y)$
Xor	$x \oplus y$	$\neg(\neg(x \wedge \neg y) \wedge \neg(\neg x \wedge y))$

Table 1. Basic logic operations with two-input AND gates and negation.

3 Optimization Rules

Our rules in Tab. 2 are size decreasing, not globally size increasing and operate on a local two-level window of the AIG before node creation. They can be categorized by three attributes: name, type of symmetry and optimization level. The AIG on which the rules operate can be *symmetric* or *asymmetric*. Symmetric means that the AIG consists of three or one conjunctions and asymmetric that it has two conjunctions. Hence the AIGs with three conjunctions in Fig. 1 are symmetric while the AIGs with two conjunctions are asymmetric. A subset of the rules is also discussed in [4].

We developed a tool which enumerates all permutation equivalent AIGs and checks if representative AIGs can be replaced by semantically equivalent AIGs. The criterion for candidate replacement AIGs is to have less nodes and that they do not cause a global node increase.

A rule consists of a one- or two-level AIG on the left-hand side (LHS) and a semantically equivalent AIG on the right-hand side (RHS). The rules are characterized by their optimization levels O1, O2, O3 and O4. Optimization level $i+1$ contains all rules of optimization level i and less. Optimization levels should be distinguished from the number of levels of the AIG simplification window, which is the height of the AIG of the LHS.

In optimization level one (O1) all optimizations in a window of size one are applied, i.e. the LHS of these rules consists of exactly one conjunction.

Starting with optimization level two (O2) the optimization window, i.e. the number of levels of the LHS, becomes two. In O2 no new node is created. The RHS is a reference to an existing sub term of the LHS or the constant $\perp$.

Optimization level (O3) allows to create at most one new node. The set of O3 rules is restricted to those two-level rules such that exactly one rule is applicable, by excluding the rules of symmetric idempotence. The assumption is that the children of the top node are already normalized, i.e. no further rule can be applied. This restriction is lifted in optimization level four (O4). While the O4 rules are clearly not confluent, we conjecture that O3 rules are confluent. The rules of O2 to O4 assume that O1 optimizations have been applied before.

Name	LHS	RHS	O	S	Condition
Neutrality	$a \wedge \top$	a	1	S	
Boundedness	$a \wedge \bot$	$\bot$	1	S	
Idempotence	$a \wedge b$	a	1	S	$a = b$
Contradiction	$a \wedge b$	$\bot$	1	S	$a \neq b$
Contradiction	$(a \wedge b) \wedge c$	$\bot$	2	A	$(a \neq c) \vee (b \neq c)$
Contradiction	$(a \wedge b) \wedge (c \wedge d)$	$\bot$	2	S	$(a \neq c) \vee (a \neq d) \vee (b \neq c) \vee (b \neq d)$
Subsumption	$\neg(a \wedge b) \wedge c$	c	2	A	$(a \neq c) \vee (b \neq c)$
Subsumption	$\neg(a \wedge b) \wedge (c \wedge d)$	$c \wedge d$	2	S	$(a \neq c) \vee (a \neq d) \vee (b \neq c) \vee (b \neq d)$
Idempotence	$(a \wedge b) \wedge c$	$a \wedge b$	2	A	$(a = c) \vee (b = c)$
Resolution	$\neg(a \wedge b) \wedge \neg(c \wedge d)$	$\neg a$	2	S	$(a = d) \wedge (b \neq c)$
Substitution	$\neg(a \wedge b) \wedge c$	$\neg a \wedge b$	3	A	$b = c$
Substitution	$\neg(a \wedge b) \wedge (c \wedge d)$	$\neg a \wedge (c \wedge d)$	3	S	$b = c$
Idempotence	$(a \wedge b) \wedge (c \wedge d)$	$(a \wedge b) \wedge d$	4	S	$(a = c) \vee (b = c)$
Idempotence	$(a \wedge b) \wedge (c \wedge d)$	$a \wedge (c \wedge d)$	4	S	$(b = c) \vee (b = d)$
Idempotence	$(a \wedge b) \wedge (c \wedge d)$	$(a \wedge b) \wedge c$	4	S	$(a = d) \vee (b = d)$
Idempotence	$(a \wedge b) \wedge (c \wedge d)$	$b \wedge (c \wedge d)$	4	S	$(a = c) \vee (a = d)$

Table 2. All locally size decreasing, globally non increasing, two-level optimization rules. "O" is the optimization level, "S" the type of symmetry. Subsumption is also known as "Absorption". The condition $a \neq b$ is a short hand for $a = \neg b$ or $b = \neg a$.

The analysis of the enumeration of all equivalence classes produced by our tool shows that Tab. 2 contains all rules that are locally size decreasing without affecting global sharing negatively. There are other locally size decreasing, but globally potentially size increasing rules, such as the two rules shown in Fig. 1. These rules should be avoided in a greedy local algorithm. For instance there are two symmetric substitution variants, Tab. 2 and Fig. 1. The latter can affect structural sharing negatively.

3.1 Experiments

We implemented our local optimization rules as part of the AIG library of the SMV2QBF tool [9]. Then we used SMV2QBF to generate AIGs for proving the correctness of simple safety properties through k-induction [13]. These AIGs are then transformed in to CNF through the standard Tseitin construction [14] and passed on to the SAT solver SateliteGTI [7,8].[1] The resulting formulas are all unsatisfiable. We used the smallest bound k for which k-induction succeeds.

		number of AIG nodes					reduction in addition to O1			
	k	O1	O2	O3	O4	Og	O2	O3	O4	Og
eijk.S298.S	58	257579	**257351**	**257351**	**257351**	260744	**0.1%**	**0.1%**	**0.1%**	-1.2%
eijk.S953.S	7	10311	**10236**	**10236**	**10236**	11791	**0.7%**	**0.7%**	**0.7%**	-14.4%
eijk.S820.S	11	18266	18111	**18071**	18091	19843	0.8%	**1.1%**	1.0%	-8.6%
eijk.S510.S	10	14519	14375	14375	**14365**	16010	1.0%	1.0%	**1.1%**	-10.3%
eijk.S832.S	11	19434	**19194**	**19194**	19215	21040	**1.2%**	**1.2%**	1.1%	-8.3%
cmu.periodic.N	96	733095	733095	**719724**	**719724**	721452	0.0%	**1.8%**	**1.8%**	1.6%
nusmv.guid7.C	27	155260	154203	**152069**	**152069**	167223	0.7%	**2.1%**	**2.1%**	-7.7%
ken.oop1.C	29	65855	64605	**63674**	**63674**	65444	1.9%	**3.3%**	**3.3%**	0.6%
nusmv.guid1.C	10	28889	28520	**27721**	**27721**	32313	1.3%	**4.0%**	**4.0%**	-11.9%
nusmv.tcas2.B	6	25999	24657	24225	**24198**	26254	5.2%	6.8%	**6.9%**	-1.0%
nusmv.tcas3.B	5	20178	19023	18644	**18618**	20409	5.7%	7.6%	**7.7%**	-1.1%
vis.prodc24.E	37	297791	289394	**257570**	**257570**	270317	2.8%	**13.5%**	**13.5%**	9.2%
vis.prodc12.E	29	204807	198235	**173352**	**173352**	183219	3.2%	**15.4%**	**15.4%**	10.5%
vis.prodc17.E	27	183883	177779	**154670**	**154670**	163807	3.3%	**15.9%**	**15.9%**	10.9%
vis.prodc15.E	23	144774	139602	**120066**	**120066**	127763	3.6%	**17.1%**	**17.1%**	11.8%
vis.prodc19.E	22	135975	131023	**112273**	**112273**	119722	3.6%	**17.4%**	**17.4%**	12.0%
texas.par2.E	2	1009	992	**813**	**813**	872	1.7%	**19.4%**	**19.4%**	13.6%
vis.prodc14.E	16	86137	82589	**69211**	**69211**	74441	4.1%	**19.7%**	**19.7%**	13.6%
vis.prodc18.E	13	64185	61385	**50755**	**50755**	54901	4.4%	**20.9%**	**20.9%**	14.5%
vis.prodc13.E	8	33849	32161	**25788**	**25788**	28195	5.0%	**23.8%**	**23.8%**	16.7%
vis.prodc16.E	5	18217	17229	**13510**	**13510**	14819	5.4%	**25.8%**	**25.8%**	18.7%

Table 3. Number of AIG nodes and node reduction.

We also implemented the original greedy optimization (Og) following [1,2,10], which calculates in advance minimum size normal forms for every two-level AIG. With a maximum of four leafs there are at most $2^{16} = 2^{2^4}$ functions. Therefore a function table and thus a function can simply be represented by a 16-bit word. The function table of a LHS can be calculated recursively. This LHS is then replaced by the normal form for its boolean function. We used a separate program that generates all two-level AIGs and picked the smallest AIG with respect to node size as normal form for all two-level AIGs implementing the same function. It turns out that only 742 boolean function can be represented by two-level AIGs.

In Tab. 3 we report on the amount of reduction that can be achieved using different optimization levels. In column 1 the name of the original SMV model from [6] is described and in column 2 the bound k. Columns 3 to 7 show the number of AIG nodes of the final formula for each of the five optimization levels. The

[1] The SAT Race 2006 showed, that SateliteGTI is still the fastest SAT solver in industrial applications together with its reimplementation in MiniSAT 2.0.

remaining four columns give the amount of reduction in number of AIG nodes achieved compared to optimization level O1. From the experiments it is clear that our rules, even though they are only local, provide considerable reduction of up to 25%, compared to the conservative approach of not enabling two-level minimization. The additional reduction obtained by using non confluent rules of optimization level O4 is miniscule. As predicted by [3] also our experiments show, that greedy local minimization (Og) [1,2,10] for AIGs is not robust. It actually often results in a considerable increase, while our approach will never increase the number of nodes.

		generation time					solving time				
	k	O1	O2	O3	O4	Og	O1	O2	O3	O4	Og
eijk.S298.S	58	**1.63**	1.68	1.66	1.65	1.92	20.58	15.64	19.54	**15.02**	22.44
eijk.S953.S	7	**0.05**	**0.05**	**0.05**	**0.05**	0.06	0.41	0.40	0.38	**0.34**	0.50
eijk.S820.S	11	**0.08**	**0.08**	**0.08**	**0.08**	0.11	**0.75**	0.82	0.85	0.83	1.54
eijk.S510.S	10	**0.06**	**0.06**	**0.06**	**0.06**	0.08	0.74	0.70	**0.69**	0.78	0.97
eijk.S832.S	11	**0.09**	**0.09**	**0.09**	**0.09**	0.11	0.95	0.93	**0.84**	0.88	1.21
cmu.periodic.N	96	**4.90**	4.92	4.91	4.91	5.12	23.80	23.83	**21.83**	**21.83**	22.25
nusmv.guid7.C	27	0.90	0.90	**0.83**	**0.83**	1.18	20.57	18.43	**13.51**	16.14	17.00
ken.oop1.C	29	0.29	0.28	**0.27**	**0.27**	0.30	**9.08**	15.40	11.88	11.90	9.34
nusmv.guid1.C	10	**0.17**	**0.17**	**0.17**	**0.17**	0.22	0.66	0.63	**0.61**	0.62	0.80
nusmv.tcas2.B	6	**0.14**	**0.14**	**0.14**	**0.14**	0.17	0.57	0.53	0.51	0.52	**0.43**
nusmv.tcas3.B	5	0.11	0.11	**0.10**	**0.10**	0.12	0.52	0.48	0.46	0.46	**0.39**
vis.prodc24.E	37	3.16	3.11	**2.75**	2.76	3.15	**19.06**	23.60	19.65	20.80	23.09
vis.prodc12.E	29	2.11	2.07	**1.82**	1.84	2.07	11.34	11.20	11.44	**10.17**	11.64
vis.prodc17.E	27	1.82	1.82	1.60	**1.58**	1.83	9.69	**9.12**	10.84	11.53	13.98
vis.prodc15.E	23	1.52	1.52	**1.29**	1.30	1.47	5.61	5.79	5.59	**5.20**	7.11
vis.prodc19.E	22	1.45	1.44	**1.23**	1.26	1.41	5.33	5.47	5.64	**4.99**	6.09
texas.par2.E	2	0.48	0.48	**0.44**	**0.44**	0.49	0.03	0.03	**0.02**	**0.02**	0.03
vis.prodc14.E	16	0.88	0.87	**0.76**	**0.76**	0.87	2.39	2.43	2.42	**2.30**	2.65
vis.prodc18.E	13	0.71	0.71	**0.61**	**0.61**	0.71	1.73	1.60	1.64	**1.56**	1.86
vis.prodc13.E	8	0.38	0.38	**0.33**	**0.33**	0.38	0.85	0.80	**0.69**	**0.69**	0.83
vis.prodc16.E	5	0.24	0.24	**0.20**	0.21	0.23	0.45	0.45	**0.36**	**0.36**	0.42

Table 4. AIG/CNF gen. time by SMV2QBF and SAT solving time by SateliteGTI.

In Tab. 4 we show the time taken to generate the CNFs on the left and on the right the time taken to prove unsatisfiability with SateliteGTI.[2] Our new rules (O3 or O4) consistently outperform conservative one-level only minimization (O1) and classical greedy optimization (Og) both with respect to generation and solving time. Here the data suggests that O4 results in more efficient SAT solving than using O3 rules alone.

[2] On a cluster of 3 GHz Pentium IV with 2 GB main memory running Debian Linux.

3.2 Conclusion

We empirically derived and classified the maximal set of rules for two-level AIG rewriting which allows a greedy local application during node generation without risking global size increase. The rule set is normalizing and of course terminating. The subset of O3 rules is conjectured to be confluent. We also identified two locally reducing but potentially globally harmful rules. Our experiments show considerable improvements in size and SAT solving time.

As future work we want to classify all rules, that potentially decrease size globally but not locally. In [3] rules are generated by a program. This approach however ignores the fact that the rules presented in this paper can always be applied greedily without search. Finally we want to apply similar ideas to multi-level AIGs [12].

References

1. P. Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT solvers. In *Proc. TACAS'00.*
2. H. Andersen and H. Hulgaard. Boolean expression diagrams. In *Proc. LICS'97.*
3. P. Bjesse and A. Borälv. DAG-Aware Circuit Compression For Formal Verification. In *Proc. ICCAD'04.*
4. R. Brummayer. C32SAT - A Satisfiability Checker For C Expressions. Master's thesis, Kepler University, Linz, Austria, February 2006.
5. R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8), 1986.
6. N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. In *Proc. BMC'03*, volume 89 of *ENTCS*. Elsevier.
7. N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proc. SAT'05*, volume 3569 of *LNCS*.
8. N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT'03.*
9. T. Jussila and A. Biere. Compressing BMC Encodings with QBF. In *Proc. BMC'06.*
10. A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE Trans. on CAD*, 21(12), 2002.
11. A. Kühlmann, M. Ganai, and V. Paruthi. Circuit-based Boolean Reasoning. In *Proc. DAC'01.*
12. A. Mishchenko, S. Chatterjee, and R. Brayton. DAG-Aware AIG Rewriting - A Fresh Look at Combinational Logic Synthesis. In *Proc. DAC'06.*
13. 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.
14. 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.