Minimizing Models for Tseitin-Encoded SAT Instances

Markus Iser, Carsten Sinz, Mana Taghdiri

Karlsruhe Institute of Technology (KIT), Germany
{markus.iser, carsten.sinz, mana.taghdiri}@kit.edu

Abstract. Many applications of SAT solving can profit from minimal models—a partial variable assignment that is still a witness for satisfiability. Examples include software verification, model checking, and counterexample-guided abstraction refinement. In this paper, we examine how a given model can be minimized for SAT instances that have been obtained by Tseitin encoding of a full propositional logic formula. Our approach uses a SAT solver to efficiently minimize a given model, focusing on only the input variables. Experiments show that some models can be reduced by over 50 percent.

1 Introduction

Many applications in logic and formal methods rely on SAT solvers as core decision procedures, and in most cases the application is not only interested in a yes/no answer, but also in a satisfying assignment (*model*) if one exists.

Models are used, for example, to represent counterexample traces in software verification, steps leading to a goal in SAT-based planning, or to build candidate conjunctions of theory atoms in SMT solving based on the DPLL(T) approach [6]. The employment of models ranges from giving information to the user—either directly or, more often, after some back-transformation to the application domain—to guiding a search algorithm when a SAT solver is used to iteratively enumerate solutions.

Minimized models try to strip off inessential information from a complete solution produced by a SAT solver. Such reduced models allow, for example, the user to focus on relevant parts of a counterexample trace, or to guide a SAT-based search process more efficiently. E.g., in DPLL(T), smaller SAT models alleviate the work of the theory solver(s), as they get passed smaller conjunctions of theory atoms; by this, the refinement loop typically needs fewer iterations.

In many cases, formulas from the application domain are not in conjunctive normal form (CNF) initially, which is, however, the input format that most SAT solvers require. Thus, they have to be transformed to CNF. A number of efficient procedures for this transformation are available [3, 9, 12, 17]. But this transformation, which typically introduces additional *encoding variables*, increases the gap between the SAT solver's solution and its interpretation in the application domain. The assignment to encoding variables is often not of interest on the application domain level.

To illustrate the problem, consider the formula $F = a \vee (b \wedge c)$, and assume that we are interested in finding a model for F. One such model would be $\{a \to 0,\ b \to 0,\ c \to 1\}$, assigning *true* to a and b, and *false* to c. This model is not minimal though, as setting a to *true* would already be sufficient to make the whole formula F true. So how can we obtain minimal models? Computing them on the CNF level is not sufficient to arrive at minimal models on the level of full propositional logic, as can be seen from our small example. When we convert it to CNF (using the Tseitin transformation), we obtain the clauses $\{\ (\bar{x}\ b)\ (\bar{x}\ c)\ (x\ \bar{b}\ \bar{c})\ (\bar{y}\ a\ x)\ (y\ \bar{a})\ (y\ \bar{x})\ (y)\ \}$, where x represents the subformula $b \wedge c$ and y the complete formula F. A minimal model of this clause set would be $\{x \to 0,\ c \to 0,\ a \to 1,\ y \to 1\}$ and, even after projecting it onto the original problem variables, we would obtain $\{c \to 0,\ a \to 1\}$, which is not the minimal model $\{a \to 1\}$ that we would like to see.

In this paper, we present algorithms that allow to compute minimal models (such as $\{a \to 1\}$ for F) efficiently, using standard SAT solvers to compute an initial (complete) model, which is then minimized. The main contribution of our paper is to also take the CNF encoding into account during minimization.[1]

2 Theoretical Background

We denote the set of propositional formulas by $\mathbb{F}$. Formulas in $\mathbb{F}$ are built from a set of variable symbols $\mathcal{V}$, operators $\{\wedge, \vee, \neg\}$, and constants $\{\top, \bot\}$. For each $F \in \mathbb{F}$, the set $\mathcal{V}_F \subseteq \mathcal{V}$ denotes the set of variables occurring in F. A *variable assignment* for a given formula F is a (possibly partial) function $\alpha : \mathcal{V}_F \rightsquigarrow \{0, 1\}$ that assigns a constant value to some variable in $\mathcal{V}_F$. We use $\mathrm{dom}(\alpha)$ to denote the set of variables for which α is defined. If $\mathrm{dom}(\alpha) = \mathcal{V}_F$, we say that the assignment is *complete*; otherwise it is *partial*. Dealing with partial assignments imposes the need for a three-valued interpretation. The interpretation of a formula F under a (partial) assignment α is denoted by $\mathcal{I}_\alpha$ and defined in Figure 1. Here, 1, 0, and U stand for *true*, *false*, and *undefined*, respectively.

We now extend the standard definition of a model to partial assignments.

Definition 1 (Model). *Given a formula F, a (partial) assignment α is a model (or a satisfying assignment) for F iff $\mathcal{I}_\alpha(F) = 1$. We use $\alpha \models F$ to denote that α is a model of F.*

In what follows, we use M_α to denote the set of *true* literals in an assignment α for a formula F. That is, $M_\alpha = \{v \mid v \in \mathrm{dom}(F) \wedge \alpha(v) = 1\} \cup \{\neg v \mid v \in \mathrm{dom}(F) \wedge \alpha(v) = 0\}$. Note that M_α uniquely defines α and vice versa.

Definition 2 (Model Minimization). *Given a model $\alpha \models F$, a model $\beta \models F$ is called* α-minimized *if $M_\beta \subseteq M_\alpha$. An α-minimized model β is* α-minimal *if no further subset $M_\gamma \subset M_\beta$ is a model of F. An α-minimal model β is* α-minimum

[1] In this paper, we specifically consider the Plaisted-Greenbaum encoding [12], but other encodings, such as the original Tseitin encoding [17], are also supported.

$$
\mathcal{I}_\alpha(\bot) = 0 \qquad\qquad \mathcal{I}_\alpha(\top) = 1
$$

$$
\mathcal{I}_\alpha(v) = \begin{cases} 1, & \text{if } \alpha(v) = 1 \\ 0, & \text{if } \alpha(v) = 0 \\ \mathsf{U}, & \text{if } v \notin \operatorname{dom}(\alpha) \end{cases} \qquad \mathcal{I}_\alpha(F \wedge G) = \begin{cases} 1, & \text{if } \mathcal{I}_\alpha(F) = 1 \text{ and } \mathcal{I}_\alpha(G) = 1 \\ 0, & \text{if } \mathcal{I}_\alpha(F) = 0 \text{ or } \mathcal{I}_\alpha(G) = 0 \\ \mathsf{U}, & \text{otherwise} \end{cases}
$$

$$
\mathcal{I}_\alpha(\neg F) = \begin{cases} 1, & \text{if } \mathcal{I}_\alpha(F) = 0 \\ 0, & \text{if } \mathcal{I}_\alpha(F) = 1 \\ \mathsf{U}, & \text{if } \mathcal{I}_\alpha(F) = \mathsf{U} \end{cases} \qquad \mathcal{I}_\alpha(F \vee G) = \begin{cases} 1, & \text{if } \mathcal{I}_\alpha(F) = 1 \text{ or } \mathcal{I}_\alpha(G) = 1 \\ 0, & \text{if } \mathcal{I}_\alpha(F) = 0 \text{ and } \mathcal{I}_\alpha(G) = 0 \\ \mathsf{U}, & \text{otherwise} \end{cases}
$$

Fig. 1. Interpretation of a formula under a (partial) assignment α.

if for each α-minimal model γ it holds that $|M_\gamma| \geq |M_\beta|$. If α is clear from the context we may write minimized instead of α-minimized, and similarly for the other terms.

Now let $\mathbb{F}_{\mathsf{cnf}} \subseteq \mathbb{F}$ denote the set of formulas in conjunctive normal form (CNF). Formulas $F \in \mathbb{F}_{\mathsf{cnf}}$ are usually represented as sets of clauses, where a clause is a set of literals. As is well known, each formula can be converted to a equisatisfiable formula in CNF, e.g., by using Tseitin's encoding.

Definition 3 (Tseitin Encoding). *Given a formula $F \in \mathbb{F}$, its Tseitin encoding, $\mathcal{T}(F) \in \mathbb{F}_{\mathsf{cnf}}$, is defined as below. Our definition uses the well-known optimization of Plaisted and Greenbaum [12].*[2]

$$
\mathcal{T}(F) = d_F \wedge \mathcal{T}^1(F)
$$

$$
\mathcal{T}^p(F) = \begin{cases} \mathcal{T}^p_{\mathsf{def}}(F) \wedge \mathcal{T}^p(G) \wedge \mathcal{T}^p(H), & \textit{if } F = G \circ H \textit{ and } \circ \in \{\wedge, \vee\} \\ \mathcal{T}^p_{\mathsf{def}}(F) \wedge \mathcal{T}^{p \oplus 1}(G), & \textit{if } F = \neg G \\ \top, & \textit{if } F \in \mathcal{V} \end{cases}
$$

$$
\mathcal{T}^1_{\mathsf{def}}(F) = \begin{cases} (\neg d_F \vee d_G) \wedge (\neg d_F \vee d_H), & \textit{if } F = G \wedge H \\ (\neg d_F \vee d_G \vee d_H), & \textit{if } F = G \vee H \\ (\neg d_F \vee \neg d_G), & \textit{if } F = \neg G \end{cases}
$$

$$
\mathcal{T}^0_{\mathsf{def}}(F) = \begin{cases} (d_F \vee \neg d_G \vee \neg d_H), & \textit{if } F = G \wedge H \\ (d_F \vee \neg d_G) \wedge (d_F \vee \neg d_H), & \textit{if } F = G \vee H \\ (d_F \vee d_G), & \textit{if } F = \neg G \end{cases}
$$

The Tseitin encoding works by introducing new propositional variables. In more detail, given a formula F, its Tseitin encoding $G = \mathcal{T}(F)$ introduces a new variable symbol d_f for each sub-formula f of F. We call the variable d_F, which

[2] Some modern implementations introduce no additional encoding variables for negated formulas and inline the negation.

stands for the complete formula, the *root variable*. The set of variables $\mathcal{V}_G$ can be partitioned into *input variables* $\mathcal{V}_G^{\mathsf{inp}}$ that stem from the original formula F and new *encoding variables* $\mathcal{V}_G^{\mathsf{enc}}$.

3 Approach

The starting point of our approach is a Tseitin-encoded formula $\mathcal{T}(F) \in \mathbb{F}_{\mathsf{cnf}}$ and a complete satisfying assignment $\alpha \models \mathcal{T}(F)$ for it, as it can be obtained by a standard SAT solver. It then computes a minimized model α' of the original formula $F \in \mathbb{F}$. To do this, it takes structural information about the partitioning of variables in $\mathcal{T}(F)$ into input variables and encoding variables into account, as well as the structural information from the Tseitin encoding.

Our minimization algorithm consists of two parts. The first works on the CNF level, and is based on a transformation of the model minimization problem to a *hitting set problem*, in which we search for a set $M_{\alpha'}$ that contains at least one literal from each clause that is assigned to true. We solve this hitting set problem by converting it to SAT, and using iterative calls to a SAT solver to obtain a minimal model α'.[3] This part is done by procedures `normalize` and `minimize` in Alg. 1. The second part, which works as a pre-processing step, exploits the structure of a Tseitin-encoded formula to further minimize the model (procedure `prune` in Alg. 1). A minimal model for the pruned formula $P \subseteq \mathcal{T}(F)$ is a *minimized* model for F that is often significantly smaller than a *minimal* model for $\mathcal{T}(F)$.

Algorithm 1: High-Level View of Model Minimization Algorithm

Input: Formula $\mathcal{T}(F)$, complete model α of $\mathcal{T}(F)$, root variable d_F
Output: Minimized model α' for F

$P = \texttt{prune}(\mathcal{T}(F), d_F)$
$N = \texttt{normalize}(P, \alpha)$
$\alpha' = \texttt{minimize}(\alpha, N)$
return α'

The three main steps of our algorithm are explained in what follows, starting with the `normalize` and `minimize` procedures that do not take information about the initial formula's structure into account.

3.1 Normalization

Given a formula $F \in \mathbb{F}_{\mathsf{cnf}}$ and a model $\alpha \models F$, the normalization step generates a problem $F' \in \mathbb{F}_{\mathsf{cnf}}$ which is an encoding of the hitting set problem mentioned above. This problem is then solved in the minimization step of the algorithm.

[3] Our main algorithm only computes an approximative solution for the hitting set problem, but in a variant of it we can also compute minimum models (see Sec. 4).

The first step of normalization, called *purification*, consists of removing irrelevant literals from F, i.e. those literals which are assigned to *false* by α.

Definition 4 (Purification). *Given a formula $F \in \mathbb{F}_{\mathsf{cnf}}$ and a model $\alpha \models F$, the purified formula $p_\alpha(F)$ is defined as follows.*

$$p_\alpha(F) = \{C \cap M_\alpha \mid C \in F\}$$

Lemma 1. *Given a formula $F \in \mathbb{F}_{\mathsf{cnf}}$ and a model $\alpha \models F$, for any assignment α' for which $M_{\alpha'} \subseteq M_\alpha$, we have $\alpha' \models F$ iff $\alpha' \models p_\alpha(F)$.*

After purification we eliminate negated literals by flipping their signs. As all literals in $p_\alpha(F)$ are pure (i.e. they occur only in one polarity), no new negations are introduced by this step, and all literals are positive afterwards.

The whole process of purification followed by flipping negated literals we call *normalization*. The formula obtained by normalization is denoted by $\nu_\alpha(F)$ and forms the basis for our minimization strategy.

3.2 Iterative Minimization

Computation of a minimal model for F is equivalent to finding a model for $\nu_\alpha(F)$ with a minimal number of *true* variables. Since we are generally only interested in models with a minimal number of input variables (i.e., from $\mathcal{V}_F^{\mathsf{inp}}$), we directly minimize assignments to these.

Minimization works by adding a version of a *cardinality constraint* to $\nu_\alpha(F)$, which starts with a bound $k = |\mathcal{V}_F^{\mathsf{inp}}|$, iteratively decreasing it, and checking by calling a SAT solver whether still a satisfying assignment with this bound exists.

Algorithm 2: Iterative Minimization

Input: Formula $F \in \mathbb{F}_{\mathsf{cnf}}$, complete model α of F, input variables $\mathcal{V}_F^{\mathsf{inp}}$
Output: Minimized model $\alpha_{\min}$ as a set $M_{\alpha_{\min}}$ of literals

$F' = \nu_\alpha(F)$, $M' = \mathcal{V}_{F'}$
repeat
  $C = \emptyset$, $E = \emptyset$, $M = M'$
  for $v \in \mathcal{V}_F^{\mathsf{inp}}$ **do**
    if $v \in M$ **then** $C = \{\neg v\} \cup C$
    else $E = \{\{\neg v\}\} \cup E$
  $(r, M') = \mathtt{solve}(F' \cup \{C\} \cup E)$
until $r = \bot$
$M^+_{\alpha_{\min}} = \{v \mid v \in M,$ and v has not been flipped by $\nu_\alpha(F)\}$
$M^-_{\alpha_{\min}} = \{\neg v \mid v \in M,$ and v has been flipped by $\nu_\alpha(F)\}$
return $M^+_{\alpha_{\min}} \cup M^-_{\alpha_{\min}}$

Algorithm 2 outlines the procedure. We use a "cardinality clause" C, which forbids assigning all k variables to true. Moreover, we remember variables already excluded from a minimal model in a set E. The SAT solver call `solve` in Line 7 is assumed to return both the satisfiability status r ($\top$ for satisfiable, $\perp$ for unsatisfiable) and a model M, if one exists. Construction of clause C ascertains that the constraint is strengthened in each iteration. Finally, we obtain a minimal model of $\nu_\alpha(F)$, which we then map back to the original problem F by taking back the variable flips that were made by the normalization procedure.

Structural Pruning. Assuming that we know that our formula $\mathcal{T}(F)$ uses an encoding like in Definition 3, and given that we also know the root variable d_F and the input variables $\mathcal{V}_F^{\mathsf{inp}}$, we can reconstruct the structure of the original formula, by recursively following the definitions of the sub-formulas of d_F until we reach a definition that is solely based on input variables.

As of Definition 3, for each subformula S of F there exists a variable d_S that is defined by clauses $\mathcal{T}_{\mathsf{def}}^p(S) \subset \mathcal{T}(F)$. It is easy to see that an encoding variable d_S has the same polarity in all its defining clauses. All literals d_X that are used to define d_S are either input variables or are themselves defined by clauses $\mathcal{T}_{\mathsf{def}}^p(X)$. Now let $\mathsf{Clauses}(l, F) = \{C \in F \mid l \in C\}$ denote all clauses in F containing the literal l. If F is clear from the context, we may simply write $\mathsf{Clauses}[l]$.

Lemma 2 (Opposite Polarity). *For all $C \in \mathcal{T}_{\mathsf{def}}^p(S)$ and all direct sub-formulas $d_X \notin \mathcal{V}_{\mathcal{T}(F)}^{\mathsf{inp}}$ of S it holds that*

$$\begin{aligned} d_X \in C &\implies \mathcal{T}_{\mathsf{def}}^p(X) = \mathsf{Clauses}(\neg d_X, \mathcal{T}(F)) \\ \neg d_X \in C &\implies \mathcal{T}_{\mathsf{def}}^p(X) = \mathsf{Clauses}(d_X, \mathcal{T}(F)) \end{aligned}$$

It follows that by parsing the defining clauses of any formula S we can recursively discover the defining clauses of its sub-formulas. Starting with the top-level Tseitin literal d_F we can thus reconstruct the syntax tree of the original formula.

The idea of structural pruning is to create a new formula $F' \subseteq \mathcal{T}(F)$ by purging all clauses that belong to definitions of sub-formulas that are not satisfied by α. Algorithm 3 outlines the procedure. We start with an empty formula (Line 1) and prepare the set of all satisfied encoding literals (Line 2). We reconstruct parts of the structure of F by following only the definitions of satisfied sub-formulas (Line 6), thus building a new formula that contains only the clauses belonging to the satisfied sub-formulas of F (Line 5).

After pruning we can normalize the new formula $F' \subseteq \mathcal{T}(F)$ and minimize α with respect to the pruned formula as shown above.

4 Experimental Results

We implemented our approach as a patch on top of MiniSAT 2.2.0 and performed the experiments on a PC (3.40GHz $\times$ 8 CPU, 8 GB Memory) running Linux (Ubuntu 12.04). Our evaluation benchmarks consist of a collection of satisfiable problems from (1) software checking problems that are shipped with the Alloy Analyzer 4 [16], (2) AIG benchmarks from SAT-Race 2010 [1], and (3) program

Algorithm 3: Structural Pruning

Input: Formula $F \in \mathbb{F}_{\mathsf{cnf}}$, model α, input variables $\mathcal{V}_F^{\mathsf{inp}}$, root encoding var. d_F
Output: Pruned formula $F' \subseteq F$

$F' = \emptyset$
$\mathsf{L} = \{l \in M_\alpha \mid \mathtt{var}(l) \in \mathcal{V}_F^{\mathsf{enc}}\}$
$\mathtt{Stack.push}(\mathsf{Clauses}[\neg d_F])$
while $C = \mathtt{Stack.pop}$ **do**
  $F' = F' \cup C$
  for $l \in C \cap \mathsf{L}$ **do**
    $\mathsf{L} = \mathsf{L} \setminus \{l\}$
    $\mathtt{Stack.push}(\mathsf{Clauses}[\neg l])$

verification problems generated by JForge [4]. In order to perform minimization, one needs to know the set of input variables of a given CNF formula, which usually occupy the first consecutive block of variable identifiers. Furthermore, in order to perform structural pruning, one needs to know the identifier of the root variable. We modified the CNF generators to produce this information as additional CNF comments (“`c input $n`”) and (“`c output $i`”), respectively.

In this section, we report on those benchmarks where at least 1% of their input variables are *don’t care*. We present the quality and performance of our approach with and without structural pruning. The results are given in Table 4. The first column gives the problem name and the second column gives the number of input variables. The next three columns give the final number of input variables, percentage of reduction (of input variables), and the runtime of our minimization approach without structural pruning. The last three columns give the results for our approach with structural pruning.

As can be seen in the table, both approaches (with and without pruning) run quickly; they actually take less than a second to perform minimization even for large CNF formulas. The quality of the results, however, differs substantially. In many cases, pruning can eliminate many more input variables without introducing much runtime overhead. This is because pruning takes advantage of the structure of the Tseitin-encoded formulas and avoids all sub-formulas whose encoding variable is a don’t-care.

Optimal Minimization. Since our iterative minimization approach does not guarantee to find a minimum assignment, we performed a second set of experiments in which we compared the outcome of iterative minimization to that of an optimal algorithm. We computed the optimal minimization using a cardinality encoding based on parallel counters [15], and iteratively calling a SAT solver to check whether the number of input variables can be reduced.

In our experiments the simpler iterative minimization approach we presented above was always able to find a minimum assignment. Moreover, its runtime turned out to be much better than the optimal algorithm (up to a factor of 168 in our tests).

		w/o Pruning			w/ Pruning		
problem	nInput	nInput	%	time (sec)	nInput	%	time (sec)
ibm18-len29-sat	983	764	22.3	0.03	575	41.5	0.04
ibm20-len44-sat	1493	1277	14.5	0.06	991	33.6	0.09
ibm22-len52-sat	2245	1932	13.9	0.11	1664	25.9	0.16
ibm23-len36-sat	1515	1308	13.7	0.06	1083	28.5	0.08
ibm29-len26-sat	362	211	41.7	0.01	134	63.0	0.02
intel-003-k-ind-30	1489	1477	0.8	0.05	1441	3.2	0.08
intel-016.aig.smv.kind-b20	27970	27833	0.5	0.49	26917	3.8	0.46
intel-019-k-ind-10	3786	3729	1.5	0.06	3587	5.3	0.10
intel-025.aig.smv.kind-b30	20399	20357	0.2	0.41	19939	2.3	0.40
intel-025-k-ind-20	13939	13895	0.3	0.50	13504	3.1	0.76
intel-032-k-ind-10	6521	6488	0.5	0.16	6188	5.1	0.24
intel-033.aig.smv.kind-b10	28428	28294	0.5	0.46	26376	7.2	0.42
itox-vc1033	57775	57040	1.3	0.37	56870	1.6	0.28
itox-vc1044	58776	58009	1.3	0.42	57822	1.6	0.29
opt-spantree Closure	673	668	0.7	0.00	201	70.1	0.00
opt-spantree SuccessfulRun	2664	2559	3.9	0.06	2559	3.9	0.07
peterson NotStuck	835	835	0.0	0.01	54	93.5	0.00
set.intersect.cegar	29497	29432	0.2	0.05	4290	85.5	0.03

Table 1. Experimental results with and without structural pruning

5 Related Work & Conclusion

Minimization of SAT models has been a research topic for many years. In literature the minimization goal usually is to reduce the number of positive literals in a model (e.g. [2, 10]). Thus the `minimize` function of Koshimura et al. [10] and ours are almost identical. However their algorithm omits the `normalization` step, which means that satisfiability according to their notion of minimality still depends on the negative literals, while our approach guarantees that all negative literals belong to don't care variables.

Other work on model minimization is often directed towards a particular application, such as model checking [11], bounded model checking [8, 13, 14], SMT solving [5] or QBF solving [7]. None of the approaches seems to work on general formulas, taking only the structural information of the CNF encoding into account as in our approach.

This paper introduced an algorithm for minimizing a given model of a CNF formula with respect to the original input variables (as opposed to the intermediate encoding variables that are introduced during CNF conversion). We transform the model minimization problem to a hitting set problem (also presented as a SAT problem), and solve it by iteratively calling a SAT solver. An optional `pruning` preprocess can be applied when structural information about the CNF encoding is available. Our experiments show that the algorithm performs well with respect to both quality and runtime. Future work that we could envisage is using our minimization approach in model counting.

References

1. SAT-Race 2010. http://baldur.iti.uka.de/sat-race-2010/, 2010. [Online; accessed 14-Feb-2013].
2. R. Ben-Eliyahu and R. Dechter. On computing minimal models. *Ann. Math. Artif. Intell.*, 18(1):3–27, 1996.
3. T. Boy de la Tour. An optimality result for clause form translation. *J. Symb. Comput.*, 14(4):283–301, Oct. 1992.
4. G. Dennis, F. S.-H. Chang, and D. Jackson. Modular verification of code with SAT. In *International Symposium in Software Testing and Analysis (ISSTA 2006)*, pages 109–120, 2006.
5. I. Dillig, T. Dillig, K. L. McMillan, and A. Aiken. Minimum satisfying assignments for SMT. In *24th Intl. Conf. on Computer Aided Verification (CAV 2012)*, pages 394–409, 2012.
6. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *16th Intl. Conf. on Computer Aided Verification (CAV 2004)*, pages 175–188, 2004.
7. E. Giunchiglia, M. Narizzano, and A. Tacchella. Clause/term resolution and learning in the evaluation of quantified boolean formulas. *J. Artif. Intell. Res. (JAIR 2006)*, 26:371–416, 2006.
8. A. Groce and D. Kroening. Making the most of BMC counterexamples. *Electr. Notes Theor. Comput. Sci.*, 119(2):67–81, 2005.
9. P. Jackson and D. Sheridan. Clause form conversions for boolean circuits. In *7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT 2004)*, pages 183–198, 2004.
10. H. F. Miyuki Koshimura, Hidetomo Nabeshima and R. Hasegawa. Minimal model generation with respect to an atom set. In *International Workshop on First-Order Theorem Proving (FTP 2009)*, pages 49–59, 2009.
11. T. Nopper, C. Scholl, and B. Becker. Computation of minimal counterexamples by using black box techniques and symbolic methods. In *Intl. Conf. on Computer-Aided Design (ICCAD 2007)*, pages 273–280, 2007.
12. D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *J. Symb. Comput.*, 2(3):293–304, 1986.
13. K. Ravi and F. Somenzi. Minimal assignments for bounded model checking. In *10th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, pages 31–45, 2004.
14. S. Y. Shen, Y. Qin, and S. Li. Minimizing counterexample with unit core extraction and incremental SAT. In *6th Intl. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI 2005)*, pages 298–312, 2005.
15. C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *International Conference on Principles and Practice of Constraint Programming (CP 2005)*, pages 827–831, 2005.
16. E. Torlak and D. Jackson. Kodkod: A relational model finder. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS 2007)*, pages 632–647, 2007.
17. G. S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125, 1970.