Extending Clause Learning of SAT Solvers with Boolean Gröbner Bases

Christoph Zengler and Wolfgang Küchlin

Symbolic Computation Group, W. Schickard-Institute for Informatics,
Universität Tübingen, Germany
http://www-sr.informatik.uni-tuebingen.de
{zengler,kuechlin}@informatik.uni-tuebingen.de

Abstract. We extend clause learning as performed by most modern SAT Solvers by integrating the computation of Boolean Gröbner bases into the conflict learning process. Instead of learning only one clause per conflict, we compute and learn additional binary clauses from a Gröbner basis of the current conflict. We used the Gröbner basis engine of the logic package Redlog contained in the computer algebra system Reduce to extend the SAT solver MiniSAT with Gröbner basis learning. Our approach shows a significant reduction of conflicts and a reduction of restarts and computation time on many hard problems from the SAT 2009 competition.

1 Introduction

In the last years SAT solvers became a vital tool in computing solutions for problems e.g. in computational biology [1,2] or formal verification [3,4]. The vast majority of SAT solvers successfully applied to real-world problems uses the *DPLL* [5] approach. DPLL is basically a complete search in the search space of all 2^n variable assignments with early cuts in the search tree when an unsatisfiable branch is detected.

It has been observed in the past that it can be profitable to enrich pure DPLL-style SAT-solving by some deduction. The extra effort of deducing new lemmas pays off when the lemmas prune the original DPLL search-tree enough to yield a net profit. However, it is clear that unrestricted deduction will choke a solver with useless clauses. This situation suggests research into suitable combinations.

Both deduction by Resolution and by the computation of Gröbner bases have been applied in the literature. The use of Resolution goes back to the original Davis-Putnam approach [5], and Buchberger's algorithm [6] has been used for theorem-proving at least since 1985 [7]. The combination approaches have been either static (in a preprocessing step) or dynamic (in the course of the search), and there are several schemes for confining deduction.

An important break-through was the development of clause learning SAT solvers (see [8, chapter 4]), which use an optimized form of resolution dynamically in conflict situations to deduce ("learn") a limited number of lemmas, so called *conflict clauses*. Resolution may deduce valuable (short) clauses as lemmas but

may blow up memory, DPLL style SAT-Solving leaves memory constant but may fail to find valuable lemmas, while BDDs, or BGBs compile an axiom system at great cost in both memory and time into an efficient canonical form for repeated use in proving theorems. Today, the dominating scheme for clause learning is *1-UIP clause learning* where a single conflict clause is derived which is guaranteed to prune the backtracking tree immediately (non-chronological backtracking).

Van Gelder [9] combined conflict driven backjumping with dynamic resolution based deduction, confined to variable-elimination resolution and binary-clause resolution, and he identified (and eliminated) equivalent literals. Bacchus [10] added binary hyper-resolution and refined the implementation. Condrat and Kalla [11] take a static approach using Gröbner basis computations in a preprocessing step. They convert all derived polynomials to clauses, but confine deduction to computationally feasible subsets of the input clauses.

In contrast, we experiment with a dynamic approach, where Gröbner basis computations are confined to conflict situations and only short polynomials are kept as clauses. We start with the clauses associated with the conflict and compile them into a suitable Boolean Gröbner basis (BGB). The idea is that the interreduction process, which is inherent in the Gröbner basis algorithm and which assures that the Gröbner basis is in (minimal) canonical form with respect to a variable ordering, will also yield short (and therefore valuable) lemmas. The approach is sound because the computation of a Boolean Gröbner basis produces only Boolean polynomials which are valid consequences of the input. Our experiments with problems from the 2009 SAT competition show that the approach is valuable on many hard problems, by yielding small clauses as lemmas which reduce conflicts and therefore save time in the DPLL search.

Clearly, both the learning of additional clauses and the use of BGBs for deduction have been investigated in the past. The approach of Condrat and Kalla [11] seems to be most similar to ours. However, they only compute BGBs before executing the DPLL solver and therefore have only static information about the problem, while we use dynamically derived conflicts. We also use a variant of polynomial algebra which is better suited for representing clauses. Dershowitz et al. [12] propose an approach for satisfiability solving based on Boolean rings. This method does not incorporate learning but relies on heavy simplification of the formula during the solving process. However, they do not compare their results with state of the art SAT solvers on real problems. In [13] they describe a method for quantifier elimination in Boolen Algebras where SAT solving is a special case. To the best of our knowledge, our specific approach proposed here is novel and merits further research.

The plan of this paper as follows: In Section 2 and Section 3 we summarize the important properties of DPLL-style SAT solving and Boolean Gröbner bases up to an extent necessary to follow the rest of the paper. Section 4 presents our approach of integrating the computation of Boolean Gröbner bases in the SAT solving process. We show extensive benchmarks of our implementation in Section 5. Section 6 finally points at some future research directions.

2 SAT Solving

We use the standard notation of propositional logic with propositional variables from an infinite set $\mathcal{V}$, Boolean operators $\neg$, $\vee$, $\wedge$, and Boolean constants "**T**" and "**F**." An *assignment* is a partial function $\alpha : \mathcal{V} \rightarrow \{0, 1\}$ mapping variables to truth values. We write $x \leftarrow b$ for $\alpha(x) = b$. vars(φ) denotes the finite set of variables occurring in a formula φ. A *conjunctive normal form* (CNF) is a conjunction of *clauses* $(\lambda_1 \vee \cdots \vee \lambda_n)$. Each *literal* λ_i is either a variable $x_i \in \mathcal{V}$ or its logical negation $\neg x_i$. It is convenient to identify a CNF with the set of all clauses contained in it. An *empty clause* is a clause where all x_i have been assigned truth values in such a way that all corresponding λ_i evaluate to 0 and therefore the whole clause is 0. It is obvious that once reaching an empty clause, no extension of the corresponding assignment can satisfy the CNF formula containing that clause. We call the occurrence of an empty clause a *conflict*. A *unit clause* is a clause where all but one x_i have been assigned, all λ_j for $j \neq i$ evaluate to 0, and therefore in order to satisfy the clause the remaining *unit variable* x_i must be assigned such that λ_i becomes 1. The process of detecting unit clauses and fixing the corresponding values for the unit variables is called *unit propagation*, which plays an important role in modern SAT solvers. Algorithm 1 summarizes the basic algorithm.

Input: C, a set of clauses
Output: `SAT` or `UNSAT`

```
1  level := 0;
2  α := ∅;
3  while true do
4      unitPropagation();
5      if C contains an empty clause then
6          level := analyseConflict();
7          if level = 0 then
8              return UNSAT
9          backtrack(level);
10     else
11         if formula is satisfied then
12             return SAT
13         level := level + 1;
14         choose an x ∈ vars(C) \ dom(α);
15         α := α ∪ {x ← 0};
```

Algorithm 1: The basic SAT solving algorithm

For each variable we save its *decision level* starting with level 1 and incrementing with each assignment. If a conflict occurs at level 0, i.e. before any assignment, the formula is obviously not satisfiable. If a conflict occurs at a higher level, the method analyseConflict() is used to compute a backtrack level and the corresponding non-chronological backtrack is performed. If no conflict is detected and

the formula is not yet satisfied, then the next variable is chosen and assigned a truth value. The method analyseConflict() is the place where the clause learning

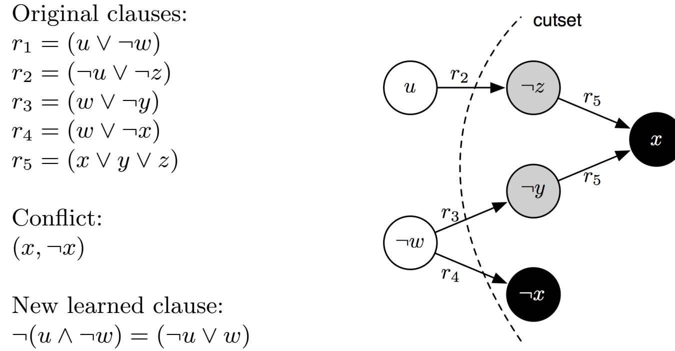

Fig. 1. An example implication graph for a given set of clauses

is actually implemented. In Figure 1 we illustrate the situation by means of an implication graph [8, chapter 4]. A node x indicates a current assignment $x \leftarrow 1$, and a node $\neg x$ indicates a current assignment of $x \leftarrow 0$. Nodes without incoming edges (white nodes) are variables which were chosen by the algorithm in line 14; they are referred to as *decision nodes*. Nodes with incoming edges (grey nodes) result from unit propagations (line 4). The clause annotated to each edge is the clause which caused the unit propagation. For example when assigning $u \leftarrow 1$ the clause $r_2 = (\neg u \vee \neg z)$ is the reason that $z \leftarrow 0$ during unit propagation. The two black nodes indicate *conflicting nodes* since from the current assignment of the variables it can be concluded that x has to be assigned 0 as well as 1. To *learn* a new clause which avoids this assignment for the future we have to find a *cut* dividing the conflicting nodes from the decision nodes. The dashed line in Figure 1 shows one example for such a cut. According to this cut we have to avoid the simultaneous assignment $u \leftarrow 1$ and $w \leftarrow 0$ in order to avoid the conflicting nodes x and $\neg x$. We thus add to our set of clauses the new clause $(\neg u \vee w)$, which is equivalent to the more natural condition $\neg(u \wedge \neg w)$. In general, new learned clauses are obtained via resolution over all clauses on the paths from conflict nodes to decision nodes.

Following the 1-UIP strategy, the backtrack level returned by analyseConflict() is determined in such a way that the new learned clause turns out to be a unit clause after backtracking to this level and therefore is used for unit propagation immediately. For more details on clause learning and different strategies to compute the cut between conflict nodes and decision nodes see [14]. After a certain number of conflicts, all assignments are taken back (learnt clauses are preserved) and the SAT solver is restarted.

3 Boolean Gröbner bases

A *Boolean Ring* $\mathcal{B}$ is a ring in which every element $a \in \mathcal{B}$ is *idempotent*, i.e. $a^2 = a$. Both $\mathcal{B}^{\oplus} = (\oplus, \wedge, \mathbf{F}, \mathbf{T})$ and $\mathcal{B}^{\leftrightarrow} = (\leftrightarrow, \vee, \mathbf{T}, \mathbf{F})$ with universes $\{0, 1\}$ are Boolean rings.[1] Let $\boldsymbol{x} = (x_1, \ldots, x_n)$. Recall that polynomials are elements of a polynomial ring $\mathcal{R}[\boldsymbol{x}]$ over a coefficient ring $\mathcal{R}$. Important rings of Boolean polynomials are $\mathcal{B}^{\oplus}[\boldsymbol{x}]$, the ring of Stone polynomials [15] in *xor normal form* XNF, and $\mathcal{B}^{\leftrightarrow}[\boldsymbol{x}]$, the ring of polynomials in *equivalence normal form* ENF. For our presentation here we consider the ring $\mathcal{B}^{\leftrightarrow}[x_1, \ldots, x_n]/\operatorname{Id}(x_1^2 + x_1, \ldots, x_n^2 + x_n)$. In contrast to the polynomial rings mentioned above, this residue class ring is a Boolean ring itself. Since the basis of the ideal $\operatorname{Id}(x_1^2 + x_1, \ldots, x_n^2 + x_n)$ is a Gröbner basis, a reduction with this basis yields unique normal forms. It is easy to see that the corresponding normal forms have a degreebound of one on every single variable. We choose these normal forms as representatives for our residue classes. For a given ordering of variables, both XNF and ENF are canonical normal forms: $A \equiv B$ if, and only if, $A^{\mathrm{XNF}} = B^{\mathrm{XNF}}$, and likewise $A^{\mathrm{ENF}} = B^{\mathrm{ENF}}$.

Let A_f be an expression denoting f. For the algebraic approach, we may convert A_f into A_f^{XNF} or into A_f^{ENF}. In our case we start with a set C of clauses, representing the conjunction $A_f = \bigwedge_{c_i \in C} c_i$. In this case it is not useful to convert the entire A_f into a single polynomial: Using Stone polynomials, all occurrences of $x_1 \vee x_2$ in the clauses must be eliminated by the transformation $x_1 + x_2 + x_1 \cdot x_2$, while the outer conjunctions lead to an analogous blow-up when using ENF-polynomials. Therefore we only convert each clause

$$c_i = (x_1 \vee \cdots \vee x_n \vee \neg y_1 \vee \cdots \vee \neg y_m) = (x_1 \vee \cdots \vee x_n \vee (y_1 \leftrightarrow \mathbf{F}) \vee \cdots \vee (y_m \leftrightarrow \mathbf{F}))$$

into a separate ENF-polynomial

$$p^{\mathrm{ENF}}(c_i) = x_1 \cdot \ \ldots \ \cdot x_n \cdot (y_1 + 1) \cdot \ \ldots \ \cdot (y_m + 1),$$

and then compute a Boolean Gröbner basis A_f^{EGB} for the set $\{p^{\mathrm{ENF}}(c_i) \mid c_i \in C\}$. Note that using Stone polynomials for the clauses (as in [7,11]) would lead to a blow-up in the polynomials prior to the Gröbner basis computation.

A_f^{EGB} is a canonical representative for the variety (the set of common roots) of the polynomials $p^{\mathrm{ENF}}(c_i)$. Hence A_f^{EGB} represents the associated Boolean function f in a minimal way, according to some ordering. In this application, we are not interested in the canonical form produced, but rather in the deduction of new polynomials in the course of the Gröbner basis computation. These are always reduced and therefore show some promise of representing small and valuable lemmas. Every new polynomial g introduced by the Gröbner basis computation must have its roots in the original variety. In logical terms, this means that $f = \bigwedge c_i \equiv \bigwedge c_i \wedge g$. Hence $\bigwedge c_i \Rightarrow g$, i.e. Gröbner basis computation produces valid lemmas [7].

[1] $\oplus$ denotes the XOR operator.

Recently, significant progress has been made in efficient custom implementations of BGBs [16]. For this initial study, however, we used a standard implementation of Buchberger's algorithm for general polynomial rings. This is possible by adding, for each variable x_i, an idempotency polynomial $x_i^2 + x_i$ to the set of clause polynomials. The mathematics for our specific approach is discussed in [17]. All general mathematical background can be found in [18,19].

4 Combining SAT Solving and Boolean Gröbner bases

In Section 2 we demonstrated how a new clause c is deduced from a current conflict by means of an implication graph. As mentioned before new learned clauses are obtained via resolution over all clauses on the paths from conflict nodes to decision nodes. In order to perform these resolutions, solvers store a reason clause $r(x)$ for each variable x implied by unit propagation. We perform learning as usual (following the 1-UIP strategy) and add for each (unsubstituted) reason clause $r(x)_i$ involved in the conflict at hand its corresponding polynomial $p^{\mathrm{ENF}}(r(x)_i)$ to a set R. After learning the 1-UIP clause, we compute a Gröbner basis $G = \mathrm{gb}(R \cup \mathcal{F})$, where $\mathcal{F}$ is the set of idempotency polynomials $x_i^2 + x_i$ for all variables x_i in $\bigcup_{p \in R} \mathrm{vars}(p)$. We collect all polynomials $p_i \in G \setminus R$ with $|\,\mathrm{vars}(p_i)| = 2$ and add their corresponding clause representation $\mathrm{clause}(p_i)$ to a set L. At the next restart of the solver, we add all clauses of L to the original clause set C. We have seen in the last section that this step does not change the semantics of the original clause set C.

Experiments have shown that current Gröbner basis packages cannot cope with large polynomial systems. Therefore we have to restrict the set R in number and length of polynomials. Instead of computing $\mathrm{gb}(R)$, we compute only $\mathrm{gb}(R')$ with $R' = \{p_i \in R \mid 2 \leq |\,\mathrm{vars}(p_i)| \leq 8\} \subseteq R$. Additionally we only perform the computation $\mathrm{gb}(R')$ when $4 \leq |R'| \leq 6$. This means we compute only Gröbner bases of subsets $R' \subseteq R$, where there are between 4 and 6 underlying clauses with $2 - 8$ literals.

We found these numbers by extensive testing. Choosing more or larger reason clauses often leads to long BGB computations and therefore slows down the overall solving process. Taking fewer or shorter reason clauses does often not produce new binary clauses and therefore does not speed up the solving process. However, we assume that these numbers strongly dependend on the chosen Gröbner basis package.

Furthermore we do not compute the Gröbner basis for every R' where the condition holds. Our tests showed that for large problems these calls to the Gröbner basis algorithm are too expensive. Therefore we only compute each 2^n-th Gröbner basis where n is the number of restarts. These two heuristics turned out to be best performing among all alternatives and therefore were chosen for the following benchmarks.

Remark 1. The additional clauses computed by the Gröbner basis algorithm could also be deduced by resolution. But there are two important points why we

use Boolean Gröbner bases at this point. First, we make use of the interreduction process, which is inherent in the Gröbner basis algorithm and which assures that the Gröbner basis is in (minimal) canonical form with respect to a variable ordering. Second, in our line of research (see also Section 6) we plan to interweave SAT solving and Boolean Gröbner bases more extensively. If one computes a Gröbner basis of clauses of more than one conflict, the result cannot be yielded by resolution anymore. At this point Gröbner bases can play off their strength.

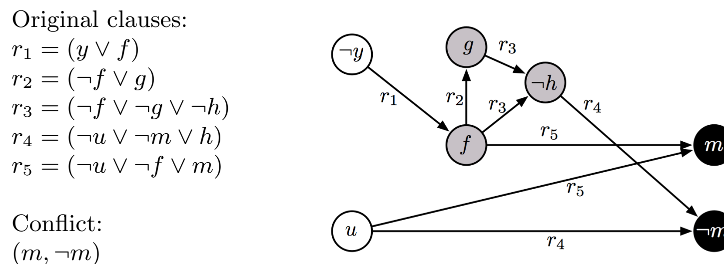

Fig. 2. An example for the Gröbner basis integration

Example 2 (Learning binary clauses with Gröbner bases). We consider a conflict with an implication graph like in Figure 2 and reason clauses $r_1, \dots, r_5$. We translate each of these clauses r_i into its corresponding Boolean polynomial $p^{\mathrm{ENF}}(r_i)$ and obtain the set $R = \{y \cdot f, (f+1) \cdot g, (f+1) \cdot (g+1) \cdot (h+1), (u+1) \cdot (m+1) \cdot h, (u+1) \cdot (f+1) \cdot m\}$. Obviously in this case $R' = R$. $\mathcal{F} = \{y^2+y, u^2+u, f^2+f, m^2+m, h^2+h, g^2+g\}$. We compute the Gröbner basis G of $R \cup \mathcal{F}$ and get $G \setminus R = \{y \cdot (u+1), y \cdot g, y \cdot (h+1), (u+1) \cdot (f+1), (f+1) \cdot (h+1)\}$. For all five polynomials p_i it holds that $|\operatorname{vars}(p_i)| = 2$. We add their corresponding clauses $(y \vee \neg u), (y \vee g), (y \vee \neg h), (\neg u \vee \neg f)$, and $(\neg f \vee \neg h)$ to L and add them to the clause set C at the beginning of the next restart. □

5 Implementation and Benchmarks

The publically available 2007 version of MiniSat[2] [20] served as a basis for our implementation *MiniSat+GB*. For the Gröbner bases computations we used the package `cgb` with lexicographical term ordering for all computations. `cgb` is implemented in the open-source Computer Algebra system Reduce[3] and it is used within the logic package Redlog [21] for various quantifier elimination procedures and for a simplifier based on Gröbner bases. So far we used `cgb` as a black box and therefore other (more efficient Boolean) Gröbner basis implementations

[2] http://minisat.se/MiniSat.html
[3] http://reduce-algebra.sourceforge.net

Table 1. Comparison between MiniSat and MiniSat+GB

		MiniSat		MiniSat+GB		% saved	
benchmark	# inst	time	conflicts	time	conflicts	time	conflicts
`aprove09`	17	62.5	129782	**51.8**	**96765**	17.1	25.4
`bioinf`	17	5245.4	5373507	**4899.4**	**5165204**	6.6	3.9
`bitverif`	12	7382.9	4857477	**5317.7**	**4126969**	28.0	15.0
`c32sat`	4	11921.9	2435435	**8539.3**	**1891845**	28.4	22.3
`crypto`	10	3167.9	408098	**2917.6**	**397982**	7.9	2.48
`palacios/uts`	6	1949.9	1179959	**1395.9**	**926299**	28.4	21.5
`parity-games`	18	2418.4	2220381	**1469.7**	**1707849**	39.2	23.1

could easily be substituted. However, with the current heuristics, we spend only about 1/1000 of the overall time in the Gröbner basis computations.

For our benchmarks we used PSL-based Reduce and version 070721 of MiniSat on one core of an Apple Mac Pro (with two 2.8 GHz Quad-Core Intel Xeon Processors). MiniSat+GB calls the C library `libreduce`, which in turn communicates via sockets with a separate Reduce process.

Table 1 presents the results. As benchmark set we chose all instances of the SAT 2009 competition[4], which could be solved by MiniSat in less than 10000 s. These are 84 instances over all. We give the name of the benchmark family, the number of instances in this set, the CPU time in seconds and the number of conflicts for both MiniSat and MiniSat+GB. The last two columns state the percentage of saved time and saved conflicts.

One can clearly see that MiniSat+GB outperforms MiniSat w. r. t. the accumulated time for each benchmark family. This is mainly because MiniSat+GB performs especially well on the large and hard instances of every family. Taking all instances of all benchmark families, we produce 13.8% fewer conflicts and therefore save 23.5% of solving time.

Figure 1 gives a diagram of all instances we benchmarked. We can observe three interesting areas: ① For instances taking less than 1 s of CPU time the overhead of computing the Gröbner bases bears no relation to the saved conflicts. Therefore our implementation always performs worse than MiniSat. ② These are instances from the `bioinf` benchmark, where we could not learn any new binary clauses at all. ③ For all instances taking more than 1000 s of CPU time MiniSat+GB clearly outperforms MiniSat (note the logarithmic scale). One of the clearest examples is the `minxorminand064` benchmark where we achieved a speedup factor of 3.3 (621.2 s vs. 2195.3 s).

6 Future Work

There are still many open questions. Are there better heuristics when to compute a Gröbner basis, based not only on the number and length of clauses? Can we profit from results [11] about the impact of term orderings? Can we learn slightly

[4] http://www.satcompetition.org/2009/

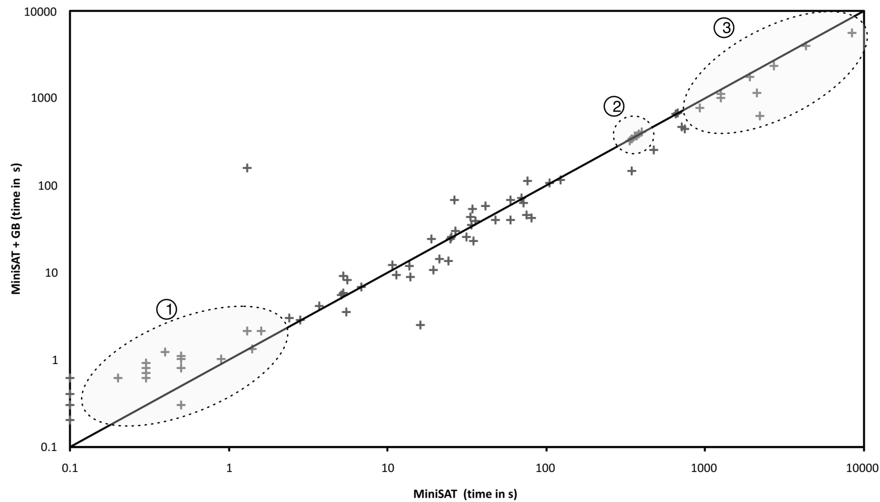

Fig. 3. Graphical comparison between MiniSat and MiniSat+GB

longer clauses that are still useful? How much improvement is possible by going to a dedicated implementation of Boolean Gröbner bases? We want to compute the Gröbner bases not only of the clauses of one conflict but also of the collected clauses of different conflicts, or of the most active clauses. These Gröbner bases could then be used to perform simplifications of the problem as described in [12]. Our approach also provides an opportunity for parallel SAT solving where one machine performs the basic SAT solving and other machines compute Gröbner bases which they communicate to the SAT solving process. On the application side we want to examine the impact of our improvements on incremental SAT solving and parametric SAT solving [22]

7 Acknowledgment

Our undergraduate student Wolfgang Braun provided valuable help with the implementation.

References

1. Lynce, I., Marques da Silva, J.P.: SAT in bioinformatics: Making the case with haplotype inference. In: Proceedings of the SAT 2006. Volume 4121 of LNCS. Springer (2006) 136–141

2. Bonet, M.L., John, K.S.: Efficiently calculating evolutionary tree measures using SAT. In: Proceedings of the SAT 2009. Volume 5584 of LNCS. Springer (2009) 4–17
3. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design **19**(1) (2001) 7–34
4. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. In Zelkowitz, M., ed.: Highly Dependable Software. Volume 58 of Advances in Computers. Academic Press, San Diego, CA (2003)
5. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5**(7) (1962) 394–397
6. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. PhD thesis, Universität Innsbruck (1965)
7. Kapur, D., Narendran, P.: An equational approach to theorem proving in first-order predicate calculus. ACM SIGSOFT Notes **10**(4) (1985) 63–66
8. Biere, A., Heule, M., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press (2009)
9. Van Gelder, A.: Combining preorder and postorder resolution in a satisfiability solver. Electronic Notes in Discrete Mathematics **9** (2001) 115–128
10. Bacchus, F.: Enhancing davis putnam with extended binary clause reasoning. In: 18th National Conference on Artificial Intelligence. AAAI Press, Menlo Park, CA, USA (2002) 613–619
11. Condrat, C., Kalla, P.: A gröbner basis approach to CNF-formulae preprocessing. In: Tools and Algorithms for the Construction and Analysis of Systems. Volume 4424 of LNCS. Springer (2007) 618–631
12. Dershowitz, N., Hsiang, J., Huang, G.S., Kaiss, D.: Boolean rings for intersection-based satisfiability. In: Logic for Programming, Artificial Intelligence, and Reasoning. Volume 4246 of LNCS. Springer (2006) 482–496
13. Seidl, A.M., Sturm, T.: Boolean quantification in a first-order context. In: Proceedings of the CASC 2003. Institut für Informatik, Technische Universität München, Garching (2003) 329–345
14. Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. JAIR **22**(1) (2004) 319–351
15. Stone, M.H.: The theory of representations for boolean algebras. Transactions of the American Mathematical Society **40** (1936) 37–111
16. Brickenstein, M., Dreyer, A., Greuel, G.M., Wedler, M.: New developments in the theory of gröbner bases and applications to formal verification. Journal of Pure and Applied Algebra **213** (2009) 1612–1635
17. Küchlin, W.: Canonical hardware representation using Gröbner bases. In: Proceedings of the A3L 2005, Passau, Germany (2005) 147–154
18. Becker, T., Weispfenning, V.: Gröbner Bases: A Computational Approach to Commutative Algebra. Springer-Verlag, New York, NY, USA (1993)
19. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms. third edn. Springer-Verlag, New York, NY, USA (2007)
20. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Theory and Applications of Satisfiability Testing. Volume 2919 of LNCS. Springer (2004) 502–518
21. Dolzmann, A., Sturm, T.: Redlog: Computer algebra meets computer logic. ACM SIGSAM Bulletin **31**(2) (1997) 2–9
22. Sturm, T., Zengler, C.: Parametric quantified SAT solving. In: Proceedings of the ISSAC 2010. ACM, New York, NY, USA (2010)