A Gröbner Basis Approach to CNF-Formulae Preprocessing*

Christopher Condrat and Priyank Kalla

Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT, USA
chris@g6net.com, kalla@eng.utah.edu

Abstract. This paper presents a CNF SAT-formulae transformation technique employing Gröbner bases as a means to analyze the problem structure. Gröbner-bases have been applied in the past for SAT; however, their use was primarily restricted to analyzing entire problems for proof-refutation. In contrast, this technique analyzes limited subsets of problems, and uses the derived Gröbner bases to yield new constraint-information. This information is then used to reduce problem structure, provide additional information about the problem itself, or aid other preprocessing techniques. Contrary to the precepts of contemporary techniques, the transformation often increases the problem size. However, experimental results demonstrate that our approach often improves SAT-search efficiency in a number of areas, including: solve time, conflicts, number of decisions, etc.

1 Introduction

The Boolean Satisfiability Problem (SAT) is formulated as finding solutions satisfying a set of Boolean equations, or to show that no such solutions exist (UNSAT). Such problems are often represented in Conjunctive Normal Form (CNF), whereby sets of literal-disjunctions (clauses) must be simultaneously satisfied through some variable assignment.

Solving for SAT-problems involves SAT-solvers. Most are based on the Davis-Putnam [1] and Davis-Logemann-Loveland [2] procedures (DPLL), which performs recursive branching and unit propagation over clauses. This technique is aided by concepts such as constraint-propagation [3], conflict analysis [4], and learning [5], which enable non-chronological backtracking [4] [6] [7], pruning the search space and reducing overall search time.

The SAT solving-time is not, however, merely a function of the variables and constraints that form the core SAT problem. Problem-representation, especially in CNF, can affect how SAT-solving performs. This is especially true when SAT-instances are transformed from system designs, for validation purposes. In automated conversion, utilities can produce "unoptimized" instances for CNF—those with constraints and variables that do not provide useful information. As a result, time and resources are wasted.

A recent area of research has therefore formed around CNF-formula transformation and simplification. This approach diverges from, or should be said, *complements*

* This work is supported, in part, by a Faculty Early Career Development (CAREER) grant from the US National Science Foundation, contract No. CCF-546859.

O. Grumberg and M. Huth (Eds.): TACAS 2007, LNCS 4424, pp. 618–631, 2007.

"classical" approaches to SAT-solving, those based on DPLL-solving [2] [4] [6] [7], by attacking the SAT problem at its source: the constraints. Algorithms and techniques such as [8] [9] [10] simplify and transform CNF-constraints through methods such as clause subsumption, hyper-resolution, and variable elimination. The applicability of such approaches varies between problems; however, most problems can benefit from at least some level of CNF preprocessing.

The goal of the preprocessor is to make the problem easier to solve, not necessarily reduce the problem size. A smaller problem implies, at the very least, that there is less the SAT-solver needs to process. However, this does not necessarily imply it is easier to solve. Indeed, some of the hardest problems, are those which have no "redundant" information present in the problem [11]. Additional information may also help clue the solver into the actual purpose of the structures in the problem represent, especially in Hybrid solvers [12]. Even the abilities of rewrite-rule-based CNF-preprocessors can be affected if constraints do not fit their simplification templates.

Preprocessing approaches have traditionally concentrated on reducing the overhead—the time needed to consider constraint information—of SAT problems prior to performing the SAT search. This "overhead" comes in the form of constraints and variables that can be represented in simpler forms, or eliminated altogether. Techniques such as Hypre [8], NiVER [9], and SatELite [10] reduce this overhead through resolution-based preprocessing.

1.1 Contemporary Preprocessing Approaches

HyPre [8] employs a form of binary reasoning, called "hyper-binary resolution," in addition to the techniques found in previous preprocessors such as 2-Simplify [13]. Hyper-binary resolution performs a resolution step involving more than two input clauses. A single size-n clause and $(n-1)$-binary clauses are resolved to a form that aids in SAT-search. HyPre's ability to resolve sets of clauses to simpler forms has been relatively successful, but at the same time it can be slow.

NiVER and SatELite use resolution to eliminate variables from a SAT-instance. Variable elimination, the older cousin to DPLL, finds itself on the other end of SAT solving from DPLL, where space, as opposed to time, increases exponentially.

Given a variable x, and two clauses, containing the variable and its negation respectively, performing resolution on x represents the following:

$$(x \vee a_1 \vee \ldots \vee a_n) \bigotimes (x' \vee b_1 \vee \ldots \vee b_m) = (a_1 \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_m) \quad (1)$$

where $\bigotimes$ is the **resolution operator**. Variable elimination is performed by resolving for a variable on all clauses that contain it as follows (for variable v):

$$\begin{aligned} C_v &= ClausesContainingLiteral(v) \\ C_{v'} &= ClausesContainingLiteral(v') \\ C_v \bigotimes_v C_{v'} &= \{c_1 \bigotimes c_2 \mid c_1 \in C_v,\ c_2 \in C_{v'}\} \end{aligned} \quad (2)$$

The variable is eliminated, but at the cost of more constraints than the original set, increasing the problem size in the general case.

NiVER [9] stands for "Non-Increasing Variable Elimination Resolution." This technique attempts to overcome the size-explosion problem associated with variable elimination by only eliminating variables in a way that does not increase problem size. Some constraint-sets resulting from variable elimination contain tautologies, which may be removed, resulting in a constraint set equal to or smaller than the original—hence "non-increasing" variable elimination.

SatELite [10] improves on NiVER by combining binary clause resolution simplification with non-increasing variable-elimination, adding new resolution rules for clause subsumption. Clause subsumption proves to be useful for simplifying clauses resulting from variable elimination, enabling an efficient clause-variable simplification procedure which can be repeated until no more reductions are possible.

2 Gröbner Bases for CNF-Transformation

This paper presents a new CNF-formula transformation approach, exploiting the power of polynomial ring algebra, particularly Gröbner bases, to transform CNF-constraints. Gröbner bases provide a computational means to derive reduced bases of sets of polynomials. The resulting polynomials better represent the solution-set and while this process can introduce new constraints and new variables, but the problem itself is simplified—easier to solve. We show that deriving extra constraint information during preprocessing may actually improve performance.

The application of Gröbner bases for SAT is nothing new. Proof-systems, such as the Nullstellensatz [14] and Gröbner/Polynomial Calculus [15] proof-systems, introduced in the mid-90s, used Gröbner bases as a means to derive proof-refutations, by generating a unit ideal from the polynomials representing the problem [16]. However, refuting an entire problem using Gröbner bases can be a time consuming, and often infeasible, task. Despite the many improvements to computation algorithms, Gröbner bases systems still have potential exponential time and space complexity, especially when analyzing large problems.

Analogous to the techniques forming the foundation of Hypre, NiVER, and SatELite, a single, partial application of Gröbner bases to a SAT instance may improve solving and problem performance. By only transforming parts of the CNF-structure, the problem, as a whole, can benefit from the reduction capabilities of Gröbner bases, while leaving alone parts which are computationally infeasible. This overall strategy fits well into a CNF-SAT transformation framework, through assisting dedicated SAT-solving tools, much as contemporary CNF-preprocessors do.

2.1 Methodology

Our CNF-transformation process is divided into phases: 1) the problem structure is first analyzed, and partitions identified; 2) CNF clauses are converted into polynomials over $\mathbb{Z}_2$; 3) the polynomials are then transformed using a Gröbner basis engine using a suitable monomial ordering derived from the partition information; 4) finally, the transformed polynomials are converted back into CNF to be used in SAT-solving. To evaluate the performance of the approach, we compare the SAT-solving results for the untransformed instances with those of the transformed versions.

A detailed explanation of the process and the concepts introduced above is the subject of the following sections.

3 Preliminaries

Boolean algebra is isomorphic to polynomial ring algebra over $\mathbb{Z}_2$. Therefore, a one-to-one mapping of operators between the two algebras is defined as follows:

$$f : B \to \mathbb{Z}_2 : \qquad (3)$$
$$\begin{array}{cc} \neg n \to n+1 & a \wedge b \to a \cdot b \\ a \vee b \to a + b + a \cdot b & a \oplus b \to a + b \end{array}$$

Addition and multiplication operations are performed $mod\ 2$ so no coefficients greater than 1 will be present. The property of idempotency ($x^2 = x$) is also present, ensuring that the polynomials will remain multi-linear.

Given a set of CNF-clauses $C = \{C_0, \ldots, C_n\}$ we can transform these clauses into a set of polynomials over $\mathbb{Z}_2$ using (Func.3):

$$C' = \{C'_0 = 1, \ldots, C'_n = 1\} \qquad (4)$$

which is equivalent to the following polynomial equations over $\mathbb{Z}_2$:

$$C' = \{C'_0 + 1 = 0, \ldots, C'_n + 1 = 0\} \qquad (5)$$

The latter form will be used throughout this paper, and for notational purposes, we will assume and omit the zero-equality ($= 0$) for the equations.

The polynomial-set generates an **ideal**—the set of all linear combinations of the polynomials—and therefore is called a **generating set** for the ideal. The individual polynomials are referred to as **generators**. In commutative algebra/algebraic geometry the set of all solutions for a set of equations is referred to as a **variety**.

What is interesting to note, is that the variety is defined, not so much by a set of polynomial equations, but rather by the *ideal* spanned by them. The effect of this is that different sets of generators can represent the same ideal, and therefore the same variety. Furthermore, some generating sets may be "better" than the original, in the sense that they are simpler or easier to solve—a better representation of the ideal. Techniques exist to find such representations. For linear equations, the well-known Gaussian Elimination technique is one such method to reduce sets of polynomials into a form which is easier to solve. However, for non-linear systems of polynomials, a generalization of this type of reduction procedure is necessary. The solution is: Gröbner bases.

3.1 Gröbner Bases

Introduced by Bruno Buchberger in 1965 [17], the theory of Gröbner bases provides a *computational framework* for deriving a special kind of generating subset of an ideal, where dividing any polynomial in the ideal by the Gröbner basis gives zero. While there are many equivalent definitions of a Gröbner basis, we choose this one as most suitable:

Definition 1. [Gröbner Basis] (From Def.1.6.1 of [18]): *For a set monomial ordering $>$, a set of of non-zero polynomials, $G = \{g_1, \ldots, g_t\}$, contained in an ideal I, is called a* **Gröbner basis** *for I if and only if for all $f \in I$ such that $f \neq 0$, there exists $i \in \{1, \ldots, t\}$ such that $lp(g_i)$ (lp = leading power-product) divides $lp(f)$.*

$$G = \textit{GröbnerBasis}(I) \iff \forall f \in I : f \neq 0, \exists g_i \in G : lp(g_i) | lp(f) \tag{6}$$

This definition of a Gröbner basis allows the basis to be derived through relatively efficient computational methods akin to deriving the greatest-common-divisor for a set of multi-variate polynomials.

Deriving a Gröbner basis generally relies on a variant of the Buchberger algorithm [17], the precise details of which can be found in [18]. The Gröbner basis begins as the initial set of polynomials to be reduced. A monomial ordering is fixed to ensure that polynomials are represented in a consistent manner (detailed in subsequent sections). Buchberger's algorithm then takes pairs of the polynomials in the basis and combines them into "S-polynomials" to cancel leading terms. An S-polynomial is defined as:

$$S(f,g) = \tfrac{L}{lt(f)} \cdot f - \tfrac{L}{lt(g)} \cdot g \quad \textit{where} \quad L = lcm(lt(f), lt(g)) \tag{7}$$

The $lt(p)$ is the *leading term* of p, and $lcm(f,g)$ is the *least common multiple* of f and g. For example, consider a set of polynomials F:

$$F = \left\{ \begin{array}{ll} f_1 = abcf + abc & f_3 = bf + f, \\ f_2 = af + f, & f_4 = cf + f \end{array} \right\} \tag{8}$$

Deriving the S-polynomial for polynomials f_1 and f_2 results in:

$$\begin{array}{rl} lcm(f_1, f_2) = & abcf \\ S(abcf + abc, af + f) = & \frac{abcf}{abcf}(abcf + abc) - \frac{abcf}{af}(af + f) \\ = & abc + bcf \end{array} \tag{9}$$

The S-polynomial is then reduced by all elements in F to a remainder r. Reduction of a polynomial p, by a single element $f_i \in F$, is denoted $p \xrightarrow{f_i} r$, and reduction by the entire set F is denoted $p \xrightarrow{F}_{+} r$. The method for reduction is multivariate polynomial division. Reducing the S-polynomial in (Eqn.9) by elements of F is shown below:

$$S(f_1, f_2) \xrightarrow{bf+f} h_1 \qquad\qquad h_1 \xrightarrow{cf+f} h_2$$

$$\begin{array}{r|l} & c \\ bf + f & abc + bcf \\ - & bcf + cf \\ \hline & abc \quad + cf = h_1 \end{array} \qquad\qquad \begin{array}{r|l} & 1 \\ cf + f & abc + cf \\ - & cf + f \\ \hline & abc \quad + f = h_2 \end{array} \tag{10}$$

No other members of F can reduce h_2 further, therefore $S(f_1, f_2) \xrightarrow{F}_{+} (abc + f)$. As a non-zero polynomial, $abc + f$ is then added to the basis F as a new element. The Buchberger algorithm has therefore discovered a new element of the Gröbner basis.

This process is repeated for all unique pairs of polynomials (including those created by newly added elements), constructing the Gröbner basis. A Gröbner basis is considered **reduced** or *minimal* if no leading terms of differing polynomials in the set divide each other and the leading-term coefficient is 1. For example, a Gröbner basis

$$G = \left\{ \begin{array}{ll} f_1 = abcf + abc & f_4 = bf + f, \\ f_2 = af + f, & f_5 = cf + f \\ f_3 = abc + f, & \end{array} \right\} \tag{11}$$

is not a *reduced* Gröbner basis, as f_1 contains a leading term that is divisible by the leading term of other polynomials. Unreduced Gröbner bases contain redundant elements, and therefore this transformation approach only works with reduced Gröbner bases.

The result of the computation is a set of polynomials that should be easier to solve than the original set, while representing the same ideal. This improved set of polynomials can then be translated to CNF form, aiding SAT-solving. This forms the basic premise of the CNF-transformation approach presented. How this is effectively implemented is the subject of the next section.

4 Transformation Process

Computing a Gröbner basis using a Buchberger-variant algorithm can exhibit exponential worst-case time and computational complexity [19], and for this reason that many different techniques have been developed [15] [18] [20] [21] [22] for improving the algorithm, often for specific applications. However, while these techniques deal with the inner-workings of algorithm itself, outside the algorithm there is less control. What control there is comes in the form of polynomial selection (i.e. partitioning) and monomial ordering. For this reason, these two aspects of the computation process are an important part of the transformation process.

4.1 Partitioning

The generators of an ideal may be separated, manipulated, and then remerged with the original set in the same manner as clauses in a CNF-instance. This can be used for parallelizing Gröbner basis computations [23] or, in this case, operating only on specific subsets of the problem.

The first stage of partitioning involves removing sets of clauses that are too computationally intensive for a Gröbner basis engine to operate on. The multi-linear form of clauses may have up to $(2^n - 1)$ monomials for n literals. Therefore large clauses are avoided. In addition, sets of small clauses, taken together, may form into very large polynomials during the process of reduction. Finally, these large and small clauses may actually be components of constraint structures represented in CNF, such as large conjunctions (logical AND)—the form of which is depicted below:

$$f = \bigwedge_{i=0}^{n-1} x_i \iff \left(f \vee \bigvee_{i=0}^{n-1} \neg x_i \right) \wedge \bigwedge_{i=0}^{n-1} (x_i \vee \neg f) \tag{12}$$

An example is:

$$f = a \wedge b \wedge c \iff (a \vee \neg f) \wedge (b \vee \neg f) \wedge (c \vee \neg f) \wedge (\neg a \vee \neg b \vee \neg c \vee f) \quad (13)$$

Directly computing for such structures can cause a large intermediate expression swell [24], and because the final form is known and easily extracted, there is no point in direct computation.

Structures such as conjunctions can be quickly isolated and removed prior to transformation. For this implementation, only conjunction-like structures were targeted (disjunctions being of similar form). Conveniently, what often remains after conjunction-removal is the "glue" that connects the conjunctions—clusters of 3-literal clauses with relatively well-connected variables, which are good candidates for transformation. These clusters of clauses can be partitioned easily by collecting all clauses associated with a single variable. This works well for industrial-type SAT-instances. However, in cases where this approach may yield clusters which are too small, repeatedly collecting all clauses which are connected to a group of clauses by two or more variables also works. This ensures that clauses are connected to the group by at least two variables, while not requiring an overly high level of connectivity which could limit applicability. This also prevents, to a great extent, random CNF structures from being clustered, which are hard for polynomial calculus [25].

4.2 Generating the Gröbner Basis

A wide variety of Gröbner basis computation-engines are available due to widespread interest in their use. Our transformation engine uses CoCoA [26], specifically its C++ library, for performing Gröbner basis reductions. The output is a reduced Gröbner basis.

The CNF-clauses are converted into polynomials using (Func.3), and converted to the form found in (Eqn.5). These polynomials form the initial generating set from which a Gröbner bases will be derived. However, if the Gröbner basis is computed directly, unexpected results may appear. Take for example a set of polynomials in $\mathbb{Z}_2$, and its corresponding Gröbner basis on the right:

$$\left\{ \begin{array}{l} abc + ac + bc + c + 1 \\ bcd + bd + cd + d + 1 \\ acd + ad + cd + d + 1 \end{array} \right\} \longrightarrow \left\{ \begin{array}{l} a + b \\ b^2d + b + 1 \\ bc + cd + c + d \\ cd^2 + bd + d^2 + d + 1 \end{array} \right\} \quad (14)$$

The second and fourth Gröbner-basis polynomials contain powered terms which should not appear due to the Boolean-ring property of idempotency. Furthermore, the monomials $d^2 + d$ maps to zero, and should not appear in the fourth equation at all.

To account for idempotency we use Fermat's Little Theorem, generalized in [27] as:

Theorem 1. [Fermat's Little Theorem] *If p is a prime number, and x is an integer, the following holds:*

$$x^p \equiv x \ (mod\ p) \quad (15)$$

The above equation can be rearranged in the following manner:

$$x^p - x \equiv 0 \ (mod\ p) \tag{16}$$

$$x^2 + x \equiv 0 \ (mod\ 2) \tag{17}$$

The above are the forms of a **vanishing polynomial** in the ring $\mathbb{Z}_p[x]$ and $\mathbb{Z}_2[x]$ respectively. The set of all such polynomial equations generates the **ideal of all vanishing polynomials** in their corresponding rings. When present during a Gröbner bases derivation, the basis elements that map to this ideal are eliminated during the reduction process.

Therefore, when using Gröbner bases for transformation, the generating set must contain a polynomial equation of form (Eqn.17) for every variable found in the generating set. The result is such that any polynomial in the resulting Gröbner basis will be in multi-linear form. Furthermore, no element of the basis will contain a redundant (i.e. zero) term.

4.3 Monomial Ordering

In the actual creation of a Gröbner basis, the results may be unimpressive: a set of polynomial equations larger than the original set, often with more monomials. While a larger basis may yield useful information, finding a smaller basis reduces the number of constraints that must be satisfied, affecting performance. One way to reduce the size of a Gröbner basis is through an efficient monomial ordering.

Monomial orders determine how the Gröbner basis is generated by affecting how leading terms—which the basis is formed upon—are ordered. The ordering-methodologies are explained in standard textbooks such as [18]. There are three general-purpose orderings: pure lexicographic (*lex*), degree-lexicographic (*deglex*), taking into account the degree of the polynomial, and a degree-reverse-lexicographic (*revdeglex*), a reversed-lexicographically ordered version of the *deglex* ordering. Our transformation approach uses a *deglex* ordering.

The *lex* / *deglex* / *revdeglex* orderings only affect the *global* monomial ordering. However, these orderings can be further refined at a *local* level by taking into account the properties of the constraint set over which the transformation is applied, notably its variables. Variables in many SAT-instances are often localized, and as a result a global variable ordering may be sub-optimal. Therefore, a variable-ordering is generated specifically for each constraint set.

During initial experiments, it was observed that the later in the variable-order a high-activity variable appeared, the smaller the Gröbner basis (activity being how often a variable appears in a set of clauses). Compared to ordering such variables first, the difference in size was often orders-of-magnitude smaller. For example, consider the following set of clauses where a is the most highly active variable:

$$(a' \vee b' \vee c) \wedge (a \vee b' \vee c') \wedge (a' \vee d' \vee e) \wedge (a \vee d' \vee e') \tag{18}$$

Using an *lex* ordering $a \prec e \prec c \prec d \prec b$ the resulting Gröbner basis is formed of three polynomials (omitting the Fermat $x^2 + x$ form elements):

$$ab + cb, \quad ad + ed, \quad edb + cdb \tag{19}$$

However, if the ordering is reversed, $b \prec d \prec c \prec e \prec a$, a different Gröbner basis is computed, one with only two polynomials:

$$de + da, \quad bc + ba \tag{20}$$

For larger sets of polynomials, the variable-ordering can have pronounced effects on the size of the resulting Gröbner bases—often by orders-of-magnitude. Our transformation approach therefore seeks to minimize the size of the Gröbner bases through an effective lexicographic ordering.

To understand how a variable order affects Gröbner bases the concepts of an **elimination ideal** and an **elimination theorem** for Gröbner basis are necessary. The consequences of these two will be described later.

Definition 2. [Elimination Ideal] From [28]: *Given* $I{=}\langle f_1, \ldots, f_s\rangle \subset k[x_1, \ldots, x_n]$, *the* i*th* **elimination ideal** I_i *is the ideal of* $k[x_{i+1}, \ldots, x_n]$ *defined by*

$$I_i = I \cap k[x_{i+1}, \ldots, x_n] \tag{21}$$

A ith elimination ideal does not contain variables $x_1, \ldots, x_i$, and neither does the basis that generates it. The basis of an elimination ideal can be a Gröbner basis by using the elimination theorem:

Theorem 2. [Elimination Theorem] From [28]: *Let* $I \subset k[x_1, \ldots, x_n]$ *be an ideal and let* G *be a Gröbner basis of* I *with respect to a lex ordering where* $x_1 \prec x_2 \prec \cdots \prec x_n$. *Then for every* $0 \leq i \leq n$, *the set*

$$G_i = G \cap k[x_{i+1}, \ldots, x_n] \tag{22}$$

is a Gröbner basis of the i*th elimination ideal* I_i.

The proof for (Thm.2) can be found in [28].

When a Gröbner basis G in variables $x_1, \ldots, x_n$ is generated using an *lex* order, the resulting Gröbner basis contains, as subsets, all Gröbner bases $G_0, \ldots, G_n$ for elimination ideals $I_0, \ldots, I_n$. As variables are progressively eliminated, the subsequent Gröbner bases containing the uneliminated variables must generate ideals in the absence of those variables. This can cause many additional elements to be generated. The Gröbner basis G can therefore have exponential size-complexity [29].

The more frequent a variable appears in a set of polynomials, and the earlier it is eliminated, the larger the Gröbner basis generally is. However, ordering higher-activity variables later in the order (accending activity) allows the variables to remain present in more elements of the Gröbner basis. The result is that it is not necessary to represent those variables—in their absence—with additional polynomials, producing a smaller basis.

Therefore, our transformation approach orders variables, on a per-set basis, by accending variable-activity. Despite the fact that a *deglex* monomial ordering is not an elimination order by nature, the Gröbner basis it generates is still reduced in overall size through this variable ordering. The computation is also fast, and the overall size of the transformed set is smaller.

The effects of variable-elimination in Gröbner bases parallels variable-elimination using resolution. In the example (Ex.18), the last polynomial in (Eqns.19) represents

the SAT-instance in the absence of the variable a. Using the resolution on (Eqns.18) to eliminate a results in:

$$C_a \bigotimes C_{a'} = \left\{ \begin{array}{l} (a' \vee b' \vee c) \bigotimes (a \vee b' \vee c') = (b' \vee c \vee c') \quad = \mathrm{T}, \\ (a' \vee d' \vee e) \bigotimes (a \vee b' \vee c') = (b' \vee d' \vee c' \vee e), \\ (a' \vee b' \vee c) \bigotimes (a \vee d' \vee e') = (b' \vee c \vee d' \vee e'), \\ (a' \vee d' \vee e) \bigotimes (a \vee d' \vee e') = (d' \vee e \vee e') \quad = \mathrm{T} \end{array} \right\} \tag{23}$$
$$= (b' \vee c' \vee d' \vee e) \wedge (b' \vee c \vee d' \vee e')$$

which is exactly equivalent to the constraints implied by the third polynomial of (Eqns.19):

$$edb + cdb = 0 \tag{24}$$

NiVER and SatELite use resolution to perform variable-elimination, while seeking to avoid the size-complexity problems associated with it. Avoiding the complexity problems associated with variable-elimination is also the goal of this approach. Our variable-ordering reduces this effect to a great degree.

4.4 Transforming Polynomials into CNF Clauses

After reduction by the Gröbner basis engine, the polynomials are converted back into CNF form using for use in SAT solving using the reverse function of (Func.3). Prior to conversion, "1" is added to each polynomial to convert the polynomials back into the form in (Eqn.4).

Final Recombination — After transforming the polynomials into CNF, the transformed clauses are merged with the original problem. The SAT-instance is then ready for solving by a SAT solver.

5 Results

The transformation technique was applied to various SAT-benchmarks. The testbench system for SAT-solving was an AMD Athlon 64 2800+ (1.6Mhz) processor with 2GB of memory, running Ubuntu Linux 6.06 x86-64. zChaff version 2004.11.15 Simplified was designated as the standard solving tool for all problems. No other preprocessing was applied for the initial set of benchmarks.

5.1 Categories of SAT-Instances

The SAT-instances used for transforming and solving were, for the most part, from industrial-category benchmarks. In other words, the benchmarks were converted from EDA designs, or other sources, into CNF for validation purposes. Proving UNSAT for such instances validates the design, whereas variants with bugs will have a SAT solution. A mix of the both SAT and UNSAT benchmarks were used. Other benchmarks include instances generated using a Bounded Model Checker [30], specifically the 12-12-barrel and longmult15 benchmarks, and a coloring benchmark.

By and large, the industrial SAT-instances dominate the results table. This is because the structure of such problems lend themselves to meaningful partitioning, where

clauses can be partitioned into well-connected sets. Furthermore, some classes of SAT instances simply do not perform well using a transformation approach which relies on reducing polynomials with each other. Random SAT instances are a good example. The paper [25] showed that random CNFs are hard for Polynomial Calculus, and though this research was carried out over non-Boolean rings, in experiments, this appears to be true for Boolean rings as well.

The problems below reflect the types of problems that can be effectively processed by the Gröbner basis transformation engine. All times are measured in seconds.

Benchmark	S	Variables		Clauses		N	PPT	Solve Time	
		Orig	Trans	Orig	Trans			Orig	Trans
12-12-barrel	U	20114	0	83619	+34752	192576	123	545	488
engine_4_nd	U	7000	+2111	67586	+6787	1269	4.09	611	574
longmult15	U	7807	+2026	24351	+8531	180	4.54	364	218
9dlx_vliw_at_b_iq1	U	24604	+2665	261473	+8675	1536	2.88	410	326
manol-pipe-c10b	U	43517	+4967	129265	+19661	453	71.38	668	621
manol-pipe-c6id	U	82022	+12873	242044	+49707	3820	236	513	504
6pipe.sat03-414	U	15800	+1266	394739	+4724	680	17.61	142	138
7pipe_q0_k	U	26512	+2174	536414	+6648	1460	3.19	235	222
8pipe_q0_k	U	39434	+2707	887706	+8324	1812	4.55	672	634
9pipe_q0_k	U	55996	+1830	1468197	+6736	1246	29.44	872	838
12pipe_bug1_q0	S	138917	+25836	4678756	+83529	15190	502	2507	1696
color-10-3-1483	S	300	+371	6475	+1593	388	4.18	115	11.86
2dlx_cc_ex_bp_f_bug1	S	171648	+43490	2614355	+139196	26094	74.83	809	367
grieu-vmpc-s05-05s	S	625	+16046	76775	+54185	6981	8.06	1010	881
grieu-vmpc-s05-27r	S	729	+34364	96849	+110258	21498	39.72	480	167

S = SAT/UNSAT; N = number of clauses processed; PPT = Preprocessing time.

Benchmark	Decisions		Net Conflicts		Implications	
	Orig	Trans	Orig	Trans	Orig	Trans
12pipe_bug1_q0	2,959,248	1,883,240	59,419	39,077	1,666,102,113	1,100,747,357
2dlx_cc_ex_bp_f_bug1	924,583	317,972	25,512	22,580	874,876,875	301,397,803
grieu-vmpc-s05-27r	264,743	69,240	152,700	34,193	32,565,075	78,017,191
color-10-3-1483	436,772	92,283	182,689	45,396	9,734,635	3,237,651

Some results have more clauses processed than are present in the original problem. In such cases, clause sets were partitioned by their connection to variables. As some clauses may have simultaneously belonged to multiple sets, this represented a form of clause "cross-fertilization" [31]. Also, the lack of additional variables in the 12-12-barrel was a result of variable elimination from polynomial equations of form:

$$a + b = 0 \quad \text{or} \quad a + b + 1 = 0 \tag{25}$$

which imply $a = b$ and $a = \neg b$, respectively.

5.2 Combining SAT Preprocessors

In addition to applying the Gröbner basis transformation technique alone, the technique was combined with another preprocessor SatELite 1.13 [10] to test how two

transformation engines complemented each other's abilities. With SatELite's ability to simplify CNF files at the clause-level, much of the overhead "clutter"created by the Gröbner basis transformations could be reduced. However, "cleanup" was not the only purpose of using SatELite: the additional constraint-information produced by the Gröbner basis transformation could be used to improve SatELite performance.

Benchmark	S	Processing Time			Solve Time			
		SatEL	GBT	Comb	Orig	SatEL	GBT	Comb
7pipe	U	41.70	3.28	45.17	333	411	382	305
9pipe_q0_k	U	70.85	29.44	113	872	594	838	539
2dlx_cc_ex_bp_f_bug1	S	194	74.83	316	809	594	367	117

SatEL = SatELite; GBT = Gröbner Basis Transformation; Comb = Combined

In all cases, the Gröbner basis engine was applied first, and then SatELite. Some benchmarks, such as the above, were able to derive benefits from a combination of preprocessors. In one instance, the solve time of the transformed problem was only less when both techniques were combined.

5.3 Interpreting the Results

The results are mixed. On many, the benefits obtained are far outweighed by the time spent during preprocessing. Furthermore, the processing varied greatly, with some problems benefiting from large numbers of clauses processed, and others very few. Also the time saved during solving varied from only marginal improvement to significant savings.

The purpose, however, is not to present a comprehensive solution, but to show the potential that lies in this approach. One stand-out result is that, despite the number of additional constraints and variables produced, in many cases the SAT solver could still find solutions in less time. Such results are encouraging, and can lead to a more refined approach, overcoming inefficiencies such as those caused by the polynomial-to-CNF translation.

6 Conclusion

We have presented a polynomial ring algebra approach to CNF-formulae transformation, using Gröbner bases as the core technique to transform polynomials. By applying this approach in an effective manner, sets of constraints, represented as polynomials can be simplified, making the problem easier to solve. We have shown that when a SAT instance is properly partitioned, and a partition-specific monomial ordering derived, this approach can perform relatively well. Also, despite the large number of clauses and variables added to the problem, extra constraint information, provided as a result of processing, can actually improve SAT solving. This technique is not fully refined; however, it shows promise and has a firm grounding in commutative ring algebra. We conclude that SAT-solving can benefit from this alternative approach to CNF-formulae transformation for preprocessing and, with additional improvements, may prove to be a viable technique for SAT preprocessing.

References

1. M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory", *Journal of the ACM*, vol. 7, pp. 201–215, 1960.
2. M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving", *in Communications of the ACM, 5:394-397*, 1962.
3. R. Zabih and D. A. McAllester, "A Rearrangement Search Strategy for Determining Propositional Satisfiability", *in Proc. Natl. Conf. on AI*, 1988.
4. J. Marques-Silva and K. A. Sakallah, "GRASP - A New Search Algorithm for Satisfiability", *in International Conference on Computer Aided Design (ICCAD)*, pp. 220–227, Nov. 1996.
5. W. Kunz and D.K. Pradhan, "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems – Test, Verification and Optimization", *IEEE Tr. on CAD*, vol. 13, pp. 1143–1158, Sep. 1994.
6. M. Moskewicz, C. Madigan, L. Zhao, and S. Malik, "CHAFF: Engineering an Efficient SAT Solver", *in In Proc. Design Automation Conference*, pp. 530–535, June 2001.
7. E. Goldberg and Y. Novikov, "BerkMin: A Fast and Robust Sat-Solver", *in Design, Automation and Test in Europe (DATE), pp 142-149*, 2002.
8. F. Bacchus and J. Winter, "Effective Preprocessing with Hyper-Resoluton and Equality Reduction", *in Proc. Intl Colloquium Automata, Languages, and Programming*, June 2003.
9. S. Subbarayan and D. Pradhan, "NiVER: Non Increasing Variable Elimination Resolution for Preprocessing SAT instances", *in International Conference on Theory and Applications of Satisfiability Testing (SAT2004)*, May 2004.
10. N. Eén and A. Biere, "Effective Preprocessing in SAT through Variable and Clause Elimination", *in International Conference on Theory and Applications of Satisfiability Testing*, June 2005.
11. E.A. Hirsch, "Random generator hgen8 of unsatisfiable formulas in CNF - SAT 2003 benchmark competition winner", `http://logic.pdmi.ras.ru/~hirsch/benchmarks/hgen8.html`
12. J. P. Warners and H. V. Maaren, "A Two Phase Algorithm for Solving a Class of Hard Satisfiability Problems", *in 90*, p. 10. Centrum voor Wiskunde en Informatica (CWI), ISSN 1386-369X, 30 1998.
13. R. I. Brafman, "A Simplifier for Propositional Formulas with Many Binary Clauses", *in Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 515–522, 2001.
14. P. Beame, R. Impagliazzo, J. Kraj'icek, T. Pitassi, and P. Pudl'ak, "Lower bounds on Hilbert's Nullstellensatz and propositional proofs", *in Proceedings of the London Mathematical Society*, pp. 73:1–26, 1996.
15. M. Clegg, J. Edmonds, and R. Impagliazzo, "Using the Gröbner basis algorithm to find proofs of unsatisfiability", *in Proc. 28th ACM Symposium on Theory of Computing*, pp. 174–183, 1996.
16. J. Buresh-Oppenheim, M. Clegg, R. Impagliazzo, and T. Pitassi, "Homogenization and the Polynomial Calculus", *Comput. Complex.*, vol. 11, pp. 91–108, 2003.
17. B. Buchberger, *Ein Algorithmus zum Aunden der Basiselemente des Restklassenringes nach einem Nulldimensionalen Polynomideal*, PhD thesis, Institute of Mathematics. University of Innsbruck, Austria, 1965.
18. W. Adams and P Loustaunau, *An Introduction to Gröbner Bases*, American Mathematical Society, 1994.
19. D. Bayer and D. Mumford, "What can be computed in algebraic geometry", *in Computational Algebraic Geometry and Commutative Algebra, Cambridge University Press, 1993, pp. 1–48, Symposia Mathematica XXXIV.*, 1993.

20. O. Bachmann and H. Schönemann, "Monomial representations for Gröbner bases computations", *in ISSAC '98: Proceedings of the 1998 international symposium on Symbolic and algebraic computation*, pp. 309–316, New York, NY, USA, 1998. ACM Press.
21. A. Giovini, T. Mora, G. Niesi, L. Robbiano, and C. Traverso, ""One sugar cube, please" or Selection Strategies in the Buchberger Algorithm", *in ISSAC '91: Proceedings of the 1991 international symposium on Symbolic and algebraic computation*, pp. 49–54, New York, NY, USA, 1991. ACM Press.
22. M. Caboara, "A dynamic algorithm for Gröbner basis computation", *in ISSAC '93: Proceedings of the 1993 international symposium on Symbolic and algebraic computation*, pp. 275–283, New York, NY, USA, 1993. ACM Press.
23. H. Shah and j. Fortes, "Tree Structured Gröbner Basis Computation on Parallel Machines", *ECE Technical Reports, Purdue Libraries*, vol. TR-EE 94-30, October 1994.
24. J. Moses, "Algebraic simplification a guide for the perplexed", *in SYMSAC '71: Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, pp. 282–304, New York, NY, USA, 1971. ACM Press.
25. E. Ben-Sasson and R. Impagliazzo, "Random CNF's are Hard for the Polynomial Calculus", *in FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, p. 415, Washington, DC, USA, 1999. IEEE Computer Society.
26. CoCoATeam, "CoCoA: a system for doing Computations in Commutative Algebra", Available at `http://cocoa.dima.unige.it`.
27. I. Niven and J. L. Warren, "A Generalization of Fermat's Theorem", *Proceedings of American Mathematical Society*, vol. 8, pp. 306–313, 1957.
28. D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms : An Introduction to Computational Algebraic Geometry and Commutative Algebra (Undergraduate Texts in Mathematics)*, Springer, July 2005.
29. D. Castro, M. Giusti, J. Heintz, G. Matera, and L. Pardo, "The hardness of polynomial equation solving", *in Found. Comput. Mathematics, 2003.*, 2003.
30. A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs", *in TACAS '99: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pp. 193–207, London, UK, 1999. Springer-Verlag.
31. N. Dershowitz, J. Hsiang, G-S. Huang, and D. Kaiss, "Boolean Ring Satisfiability", *in International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, May 2004.