Boolean Ring Satisfiability

Nachum Dershowitz[1,⋆], Jieh Hsiang[2], Guan-Shieng Huang[3,⋆⋆], and Daher Kaiss[1]

[1] School of Computer Science, Tel Aviv University, Ramat Aviv, Israel, {nachumd,daherk}@tau.ac.il
[2] Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan, hsiang@csie.ntu.edu.tw
[3] Department of Computer Science and Information Engineering,
National Chi-Nan University, Puli, Nantou, Taiwan, shieng@ncnu.edu.tw

Abstract. We propose a method for testing satisfiability based on Boolean rings. It makes heavy use of simplification, but avoids the potential size increase associated with application of the distributive law by employing a combined linear and binomial representation. Several complexity results suggest why the method may be relatively effective in many cases. The framework is also amenable to learning from intersections, as in Stålmarck's method. Some experiments have been undertaken.

1 Introduction

Simplification has been used successfully in recent years in the context of theorem proving. This process is based on a well-founded notion of "simplicity", under which one can delete intermediate results that follow from known (or yet to be derived) simpler facts. Simplifying as much as possible at each stage can greatly reduce storage requirements. Simplification-based theorem-proving strategies, as in the popular term-rewriting approach, have been used to solve some difficult problems in mathematics, including the long-open Robbins Algebra Conjecture [12]. For term rewriting in automated deduction, see [6].

A natural way of incorporating simplification in propositional reasoning is to use the *Boolean-ring* (BR) formalism. Boolean rings obey the following identities:

$\boldsymbol{xx = x}$	$\boldsymbol{x0 = 0}$	$\boldsymbol{x1 = x}$
$\boldsymbol{x+x = 0}$	$\boldsymbol{x+0 = x}$	$-x = x$
$xy = yx$	$(xy)z = x(yz)$	
$x + y = y + x$	$(x + y) + z = x + (y + z)$	$\boldsymbol{x(y+z) = xy+xz}$

where (nilpotent) + is *exclusive-or* and (idempotent) juxtaposition is *logical-and*, $\wedge$. (The additive inverse $-x$ is useless.) It is straightforward to express inclusive-or and negation: $x \vee y$ is $xy + x + y$ and $\bar{x}$ is $x + 1$.

The Boolean-ring formalism differs from Boolean algebra in that it defines a unique normal form (up to associativity and commutativity of the two operators) for every Boolean formula, called a *Boolean polynomial* (also known as a *Zhegalkin polynomial* or *Reed-Muller normal form*). Using Boolean rings, however, is not without drawbacks: Applying the distributive law (the bottom-right equation) can cause the length of a Boolean polynomial to grow exponentially in the worst case.

This paper focuses on satisfiability testing using Boolean rings (which is NP-hard [2]). Several possibilities are outlined in the next two sections. Section 4 proposes a Davis-Putnam-like method for satisfiability. We use a novel representation, separating the set of formulæ into two parts (linear, binomial), each of which, on its own, can be dealt with efficiently. This is followed (in Section 5) by some suggestions for practical improvements. Section 6 provides several new relevant complexity results. We have NP-completeness, which immediately implies completeness of the proposed method. We also give polynomial-time results for restricted classes of Boolean-ring formulæ corresponding to the two parts mentioned above. These results provide justification as

⋆ Research supported in part by the Israel Science Foundation (grant no. 254/01).
⋆⋆ Research supported by the National Science Council of the Republic of China, Taiwan (grant no. NSC92-2213-E-260-012).

to why the proposed method may be efficient. Then (in Section 7) we apply the Boolean-ring representation in a framework—akin to Stålmarck's method [14] and recursive learning [11]—that includes computing the intersection of sets of formulæ. Preliminary experimental results may be found in Section 8. This is followed by some brief comments.

2 Satisfiability

There are three basic approaches to deciding satisfiability:

Normalize. To decide satisfiability, one can transform a formula into a normal form that clearly distinguishes between satisfiable and unsatisfiable cases. Such canonical representations include disjunctive normal form (DNF) and conditional normal form (as in OBDDs). The Boolean-ring normal form (BNF), described above, can be obtained directly by applying the emboldened ring axioms from left to right to any formula [7]. Tautologies reduce to 1; contradictions, to 0; contingent formulæ, to neither. But any normal form can of necessity be exponentially larger than the original. For example, the BNF of $(p+p')(q+q')(r+r')$ is $pqr+pqr'+pq'r+pq'r'+p'qr+p'qr'+p'q'r+p'q'r'$. In the process of normalization, terms that appear twice in a sum cancel each other out (since $x + x = 0$), but this method is still, in general, impractical.

An alternative [4, 8] is to construct a Gröbner basis (confluent and terminating rewrite system) from the initial set of equations, plus idempotence ($xx = x$) for each propositional variable (x). To begin with, rather than present the whole given formula as one big equation, conjunctions $AB = 1$ are divided into two, $A = 1$ and $B = 1$, and inclusive disjunctions $A \vee B = 0$ into $A = 0$, $B = 0$. The BR axioms are applied at each step, and equations are "inter-reduced" by using each as a simplifier of others. The resultant reduced set of Boolean polynomials is unique up to associativity and commutativity of sums and products. By imposing an ordering on (variables and extending it to) monomials, a unique normal form for Boolean functions is obtained. Efficient techniques (congruence closure, Gröbner bases) can be applied to the generation task. A structure-sharing ("decision diagram") exclusive-or normal from is presented in [9].

Search. The Davis-Putnam-Logemann-Loveland (DPL) procedure [5] was the first attempt to solve the satisfiability problem efficiently, by employing a backtracking search strategy, plus heuristics to minimize formulæ. A truth value is assigned to a variable and the formula is recursively solved, trying another value when no solution is found. There are many fast implementations today based on this old procedure.

In the same vein, one can easily represent formulæ as sets of Boolean polynomials (without any exponential blowup in size), and use a similar backtracking method. See [8]. Splitting is done on variables, as usual, but formulæ are kept as simplified BR equations. Section 4 below improves on this by employing a new BR representation.

Saturate. The traditional mechanical approach to satisfiability of a propositional formula is to refute its negation and try to infer a contradiction. The total number of consequences one needs to derive is, of course, in general, exponential. Most nontrivial proofs require some form of case splitting. In Section 7, we combine limited saturation with search, within a BR framework.

3 Simplification

To counter the certain exponential cost of naïve realizations of satisfiability methods, simplification at intermediate stages is of paramount importance. Database entries are replaced with simpler ones (in some well-founded sense) by making *polynomial-time* inferences and deleting now-redundant formulæ. Such steps reduce the likelihood of suffering from the potentially exponential aspect of the approach (be it case analysis, splitting, merging, or distribution).

Regardless of the method, it is helpful to make cheap and valuable deductions—which may enable additional simplifications—as early as possible. In particular, virtually all approaches employ some mechanism for detecting "necessary" assignments. Simplification rules used in DPL and OBDD provers [3] include tautology, unit (BCP), pure literal, subsumption, and failed literal. (In practice, tautology and pure-literal are often omitted.) In the search approach, after assigning 0

or 1 to a variable, simplifiers like $x0 = 0$, $x1 = x$, $x \vee 0 = x$, and $x \vee 1 = 1$ should be applied, regardless of the specific manner in which formulæ are represented, since these rules may result in the deletion of all occurrences of some other variables. Simplification was used in [13] to preprocess CNF formulæ coming from circuit testing, and other sources, and found to reduce the number of variables to 1/3 in many cases. In a normalization approach, the same simplifiers can dramatically reduce the size of formulæ. The first five Boolean-ring axioms are simplifiers. They can be easily implemented by keeping terms sorted.

4 Representation

We argue that Boolean rings are an especially convenient framework for simplification. As mentioned already, distributivity is the potentially expensive step (even when directed acyclic graphs are used for shared subterms). We propose a new representation that circumvents this problem.

A *linear equation* (over $\mathbf{Z}_2$) is a Boolean equation in which each monomial is either a single variable or a constant; it takes the form $x_1 + \cdots + x_n = 1$ or $x_1 + \cdots + x_n = 0$, where the x_i are distinct propositional variables. A *binomial equation* is a Boolean equation with at most two monomials, that is, an equation of one of the three forms: $P = Q$, $P = 0$, or $Q = 1$, where P and Q are products of distinct propositional variables (distinct on account of idempotence). Simple equations, $x = 0$, $y = 1$, or $x = y$, for propositional variables x, y, as well as degenerate equations, $0 = 0$, $1 = 1$, or $1 = 0$, will be considered both linear and binomial.

Let $\mathcal{B}$ be a set of binomial equations and $\mathcal{L}$ be a set of linear equations over the propositional variables. Instead of solving a general set of Boolean-ring equations (e.g. $xy + x + y = 0$ implies $x = 0$ and $y = 0$), we will decide the satisfiability of $\mathcal{B} \cup \mathcal{L}$. Simplification by $\mathcal{B}$ of $\mathcal{L}$, and vice-versa, will be severely limited.

Given a system $\mathcal{R} = \mathcal{B} \cup \mathcal{L}$ and an ordering $>$ on monomials, inference proceeds as follows:

1. *Termination Test.* If $1 = 0$ has been inferred, the system is unsatisfiable.
2. *Tautology Deletion.* Remove all trivial equations $A = A$.
3. *Decomposition.* Decompose any equation $x_1 x_2 \cdots x_k = 1$ in $\mathcal{B}$ into $x_1 = 1, \ldots, x_k = 1$.
4. *Unit Rule.* Use all unit equations of the form $x = 0$ or $x = 1$ in $\mathcal{R}$ to simplify equations in $\mathcal{R}$.
5. *Equivalence Rule.* If two variables are equated, $x = y$, replace one by the other throughout $\mathcal{R}$.
6. *Simplification.* Inter-reduce one equation by another within $\mathcal{B}$ and within $\mathcal{L}$.
7. *Splitting.* Split the system of equations by considering, $\mathcal{R} \cup \{x = 1\}$ and $\mathcal{R} \cup \{x = 0\}$, individually and recursively, for some propositional variable x appearing in $\mathcal{R}$.

Equations in $\mathcal{B}$ and $\mathcal{L}$ are processed independently, except for the *Unit Rule* and *Equivalence Rule*. Splitting on x and applying the unit rule eliminates x from both $\mathcal{B} \cup \{x = 1\} \cup \mathcal{L}$ and $\mathcal{B} \cup \{x = 0\} \cup \mathcal{L}$. One advantage of this mixed representation is that relatively fast methods exist for processing each of the two components (see Section 6). The *Simplification* step is not needed for completeness, but rather to improve search efficiency. The choice of ordering $>$ is flexible: It may prefer short monomials, to keep equations short, or long ones, to maximize the likelihood of a simplifier applying.

The method presented above is complete for propositional reasoning (Theorem 3 below). Software is available to handle the computations within each of the two sets, $\mathcal{B}$ and $\mathcal{L}$. Distributivity is not needed when simplifying Boolean terms, because equations in $\mathcal{L}$ are not used to simplify $\mathcal{B}$, so formulæ do not grow very large, as in more conventional Boolean-ring based methods.

Example 1. Suppose we want to prove the validity of the formula

$$(p \vee q) \wedge (\bar{q} \vee s) \wedge (\bar{s} \vee p) \wedge (\bar{p} \vee r) \wedge (\bar{r} \vee \bar{p} \vee t) \Rightarrow (t \wedge r).$$

In order to represent the clause $p \vee q$, a new variable u for $\bar{p}$ is required. The clausal form of its negation, written as Boolean equations, is as follows:

$$(1)\ qs = q \quad (2)\ sp = s \quad (3)\ prt = pr \quad (4)\ rt = 0 \quad (5)\ p + u = 1 \quad (6)\ qu = u \quad (7)\ pr = p$$

Eq. (4) simplifies (3) to $pr = 0$, which, in turn, simplifies (7) to $p = 0$. This invokes the *Unit Rule*, and (2,5) become $s = 0$ and $u = 1$, which simplify (1) to $q = 0$. Finally, $u = 1$ and $q = 0$ simplify (6) into the contradiction $1 = 0$, concluding the proof (*sans* splits). □

5 Cross Fertilization

It is advantageous to allow some "cross fertilization" between $\mathcal{B}$ and $\mathcal{L}$, so that useful equations can migrate from one set to the other. Consider a linear equation $x + y + z = 1$. By assigning 1 to variables x and y, the equation reduces to $z = 1$, since the even number of 1's cancelled each other. Thus, we may conclude that the binomial equation $xyz = xy$ is a logical consequence. In general, a linear equation $y_1 + y_2 + \cdots + y_k = c$, implies the binomial $y_1 y_2 \cdots y_k = 0$, if either k is even and $c = 1$, or k odd and $c = 0$. In the remaining two cases, it implies k binomials $y_1 y_2 \cdots y_k = y_1 y_2 \cdots y_{k-1}$. We emphasize that *every* linear equation can breed some binomial equation(s). If we have $w + x + y = 0$ and $x + y + z = 1$ in $\mathcal{L}$, we can add $wxy = 0$, $xyz = xy$, $xyz = xz$, and $xyz = yz$ to $\mathcal{B}$. More interestingly, the sum of the two linear equations is $w + z = 1$, which implies $wz = 0$.

Additional simplifications within $\mathcal{B}$ are also possible. Suppose we have $MP = QR$ and $MQ = 0$, where M, P, Q, R are monomials. Since either M or Q equals 0, it is evident that either MP or QR equals 0. However, $MP = QR$; hence both must be 0. Therefore, we can use $MQ = 0$ to reduce $MP = QR$ to two equations, $MP = 0$ and $QR = 0$. This is an example of a critical pair computation [6], which results—after simplification—in smaller equations. In general, entailment of equations in either $\mathcal{B}$ or $\mathcal{L}$ alone is cheap (cf. Theorems 1, 2 below).

In practice, it is better when short equations are derived. A shorter equation has more chance to simplify others, and after performing simplification, the shorter equations may be retained more economically.

6 Completeness and Complexity

In this section we present several complexity results, including a reduction from SAT to the corresponding problem in Boolean rings (BRSAT), and two polynomial-time subclasses (for $\mathcal{B}$ and $\mathcal{L}$). Completeness of the whole method follows as a direct consequence of NP-completeness. The other results give some evidence as to why our method may be relatively effective.

Let Linear-BRSAT be the problem of solving a system of linear equations over the Boolean ring. Gaussian elimination can solve Linear-BRSAT in $\mathcal{O}(n^{2.376})$ time:

Theorem 1. Linear-BRSAT *is solvable in cubic time.*

Note that, although Tseitin formulæ have an exponential lower bound for resolution [15, 1], they can be solved easily within $\mathcal{L}$ alone.

Let Binomial-BRSAT be the problem of solving a system of binomial equations. The well-known Horn-SAT case falls within this class, since any Horn clause $x_1 \wedge \cdots \wedge x_m \rightarrow y$ is equivalent to $x_1 \cdots x_m y = x_1 \cdots x_m$ in the Boolean ring. In fact, Binomial-BRSAT defines the same Boolean functions as does Horn-SAT. Thus:

Theorem 2. Binomial-BRSAT *is linear-time solvable.*

Full computation of the Gröbner basis of $\mathcal{B}$ is not feasible, since its size can be exponential in the number of variables (though we have not determined the precise complexity of computing an efficient representation of the basis).

Despite the fact that Linear-BRSAT and Binomial-BRSAT are easy, their combination, BRSAT (satisfiability over $\mathcal{B} \cup \mathcal{L}$) is NP-complete, as one would expect. Recall that a clause $l_1 \vee l_2 \vee \cdots \vee l_m$ can be easily converted into an equation by expanding $(l_1 + 1)(l_2 + 1) \cdots (l_m + 1) = 0$. If there are more than two positive literals amongst the l's, the final equation will contain at least four monomials. To avoid this, we introduce new Boolean variables that serve as the *complement* of the positive literals. Suppose l_i is a positive literal, a new variable $\bar{l_i}$ and new equation $l_1 + \bar{l_i} = 1$ are introduced, and $l_i + 1$ is replaced by $\bar{l_i}$. Finally, several linear equations and one binomial equation represent the clause.

Theorem 3. BRSAT *is NP-complete.*

This theorem directly implies the completeness of the method. This is because splitting alone is complete for propositional calculus, and the reduction from SAT (in the proof of this theorem) implies that every Boolean formula can be converted into Bin-Lin form.

7 Intersection Method

The innovative approach of Stålmarck's Prover [14] has been used successfully for testing satisfiability of several large-scale industrial problems. The core idea is to learn as much as possible from case splits. Such saturate-by-intersection methods (also [11]) proceed via iterative deepening:

1. Perform cheap (i.e. polynomial) inferences. Check if done.
2. Choose a variable to split on; recur on each case.
3. If either case succeeds, then done; otherwise, merge results.
4. If something learned, add it to formulæ and reset the list of variables.
5. If no unsplit variables remain, increase depth up to current bound; otherwise, continue at same depth.
6. Repeat with incremented bound on maximum depth.

By "merging", we mean looking for consequences that hold in both cases. Since the process is repeated whenever new formulæ are learned, the search can involve a much larger than necessary total number of splits.

Formulæ in [14] are represented as "triplets", of the form $x \Leftrightarrow y \wedge z$, where x, y, z are propositional variables, which is just the binomial $x = yz$ in BR. We have implemented such a scheme for the full Bin-Lin representation (see the next section). The low complexity of inference for either binomials or linear equations means that one can tractably test for small shared simplifiers, even if they do not appear explicitly in $\mathcal{B}$ or $\mathcal{L}$.

8 Results

The degree to which simplification and learning can sometimes reduce the number of splits needed in a backtrack search is the main practical question. In our initial experiments with an implementation of a naïve Boolean-ring-based search method (of Section 2), the number of splits was reduced by 30% [10]. This saving, however, came at the price of time-consuming simplification, mainly because—in that implementation—distributivity was needed, which is not the case with the method proposed here.

We implemented the intersection method of the previous section (in about 5K lines of C^{++}), using the Bin-Lin representation of Section 4, and incorporating conflict learning (but no other DPL improvements). We ran the program on 700 real-life hardware verification examples and observed a reduction of about 30% in the number of the splits when intersections were computed. Table 1 displays some representative results. Despite the heavy cost of merging, there was an overall average reduction of 3% in run time, since merging often produces simplifiers that greatly reduce the need for splitting. Although our experiments to date included only 5–10 input variables, the number of intermediates sometimes exceeded 100, due to the need for temporary variables to convert to the Bin-Lin representation. The code wasn't designed to be competitive in terms of performance; thus constant propagation over the Bin-Lin system was noticeably time consuming. (For this reason, we could not perform larger tests within the allotted 5-minute timeout.)

Table 1. Representative runs with and without merging

Variables		Splits		Reduction
Input	Temp.	Plain	Merge	
10	103	3524	1780	49.4%
10	60	4894	2820	42.4%
9	60	3258	2951	9.4%
9	60	3564	2196	38.4%
8	36	2050	547	73.3%
8	77	332	332	0.0%
7	80	486	349	28.2%

Variables		Splits		Reduction
Input	Temp.	Plain	Merge	
7	64	243	210	13.6%
7	58	163	197	-20.9%
6	32	121	132	-9.1%
6	36	108	66	38.4%
6	31	127	154	-21.3%
5	24	41	35	13.6%
5	27	62	38	38.7%

In these tests, as the complexity of the problem increased, the reduction in splits improved. This trend can be seen in the above table, and may be explained by the higher probability of obtaining new facts in the intersection.

9 Discussion

Simplification is more time-consuming than splitting on a variable or evaluating a truth-assignment. Splitting a variable can usually be carried out in linear time, but we know of no linear-time algorithm for checking or performing simplification. (A polynomial simplification algorithm is straightforward, simply trying all possible pairs of input equations.)

Besides the ease of incorporating simplification, Boolean rings are a suitable representation for preprocessing for the following reason: Let $\mathcal{C}$ be the set of Boolean formulæ over all binary and unary operations, $\mathcal{D}$ be over $\{\vee, \wedge, \bar{}\}$, and $\mathcal{G}$ over $\{1, +, \wedge\}$. As shown in [9], any formula in $\mathcal{C}$ is linearly reducible to one in $\mathcal{G}$, but may not be linearly reducible to one in $\mathcal{D}$. Hence, Boolean-ring formulæ can preserve the structure of *any* Boolean formulæ, while Boolean algebra requires additional variables. This ability to preserve structure is also important for Stålmarck's method (cf. Section 7), which is sensitive to the structure of the input formula.

Nilpotence of + makes it possible to express parity succinctly as a linear equation in the Boolean ring. This feature allows for very simple BR proofs of the pigeon-hole principle and the mutilated checkerboard. In contrast, in Boolean algebra, the shortest corresponding formula is of quadratic length, if no new variables are introduced. Introducing new variables normally increases computational effort.

It should also be interesting to identify additional subclasses of Boolean rings (like Binomial-BRSAT) for which satisfiability testing can be accomplished by simplification alone, without splitting.

References

1. Ben-Sasson, E., and Wigderson, A.: Short proofs are narrow — resolution made simple. Journal of the ACM **48** (2001) 149–169
2. Bloniarz, P. A., Hunt, H. B., III, and Rosenkrantz, D. J.: Algebraic structures with hard equivalence and minimization problems. Journal of the ACM **31** (1984) 879–904
3. Bryant, R. E.: Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys **24** (1992) 293–318
4. Clegg, M., Edmonds, J., and Impagliazzo, R.: Using the Groebner basis algorithm to find proofs of unsatisfiability. Proc. 28th ACM Symposium on Theory of Computing (1996) 174–183
5. Davis, M., Logemann, G., and Loveland, D.: A machine program for theorem proving. Communications of the ACM **5** (1962) 394–397
6. Dershowitz, N., and Plaisted, D. A.: Rewriting. Handbook of Automated Reasoning, Robinson, A., and Voronkov, A., eds., vol. 1, chap. 9, Elsevier (2001) 535–610
7. Hsiang, J., and Dershowitz, N.: Rewrite methods for clausal and non-clausal theorem proving. Proc. 10th Intl. Colloquium on Automata, Languages and Programming (Barcelona, Spain), Lecture Notes in Computer Science **154**, Springer-Verlag (1983) 331–346
8. Hsiang, J., and Huang, G. S.: Some fundamental properties of Boolean ring normal forms. DIMACS Series in Discrete Mathematics and Theoretical Computer Science **35** (1997) 587–602
9. Hsiang, J., and Huang, G. S.: Compact representation of Boolean formulas. Chinese Journal of Advanced Software Research (1999)
10. Jan, R. L.: Experimental results on propositional theorem proving with Boolean ring. Master's thesis, Department of Computer Science and Information Engineering, National Taiwan University (1997)
11. Kunz, W.: Recursive learning: A new implication technique for efficient solutions to CAD problems — test, verification, and optimization. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **13** (1994) 1143–1158
12. McCune, W.: Solution of the Robbins problem. Journal of Automated Reasoning **19** (1997) 263-276
13. Marques-Silva, J. P.: Algebraic simplification techniques for propositional satisfiability. Technical Report RT/01/2000, INESC (2000)
14. Sheeran, M., and Stålmarck, G.: A tutorial on Stålmarck's proof procedure for propositional logic. Proc. 2nd Intl. Conference on Formal Methods in Computer-Aided Design, Lecture Notes in Computer Science **1522**, Springer Verlag (1998) 82–99
15. Urquhart, A.: Hard examples for resolution. Journal of the ACM **34** (1987) 209–219