# Boolean Rings for Intersection-Based Satisfiability

Nachum Dershowitz[1,⋆], Jieh Hsiang[2,⋆⋆],
Guan-Shieng Huang[3,⋆⋆⋆], and Daher Kaiss[4]

[1] School of Computer Science, Tel Aviv University,
Ramat Aviv, Israel, nachumd@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, Nantou, Taiwan, shieng@ncnu.edu.tw
[4] Design Technology Solutions Group, Intel Corporation,
Haifa, Israel, daher.kaiss@intel.com

> There is not a person in this courtroom
> who has never told a lie,
> who has never done an immoral thing,
> and there is no man living
> who has never looked upon a woman
> without desire.[1]
>
> —Harper Lee: *To Kill a Mockingbird*

**Abstract.** A potential advantage of using a Boolean-ring formalism for propositional formulæ is the large measure of simplification it facilitates. We propose a combined linear and binomial representation for Boolean-ring polynomials with which one can easily apply Gaussian elimination and Horn-clause methods to advantage. We demonstrate that this framework, with its enhanced simplification, is especially amenable to intersection-based learning, as in recursive learning and the method of Stålmarck. Experiments support the idea that problem variables can be eliminated and search trees can be shrunk by incorporating learning in the form of Boolean-ring saturation.

## 1 Introduction

Simplification has been used successfully in recent years in the context of theorem proving. This process requires a well-founded notion of “simplicity”, under

⋆ Research supported in part by the Israel Science Foundation (grant no. 250/05).

⋆⋆ Research supported in part by the National Science Council of the Republic of China, Taiwan (grant no. NSC94-2213-E-002-011).

⋆⋆⋆ Research supported by the National Science Council of the Republic of China, Taiwan (grant no. NSC94-2213-E-260-016).

[1] Atticus presumably did not mean $\neg\exists m.\neg\exists t,w.\neg \mathit{Desired}(t,m,w)$. Bill Clinton was reported to have used a similar triple negation.

which one can delete intermediate results that follow from known (or yet to be derived) simpler facts, without jeopardizing completeness of the inference mechanism. 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 [19]. Term rewriting means using equations asymmetrically to replace "equals by equals" in the direction that decreases the complexity of the term (according to some well-founded measure). For term rewriting as a tool in automated deduction, see [1, 12]. For an abstract theory of inference with simplification, see [11, 5].

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

$$\boxed{\begin{array}{ccc}
\boldsymbol{xx = x} & \boldsymbol{x0 = 0} & \boldsymbol{x1 = x} \\
\boldsymbol{x{+}x = 0} & \boldsymbol{x{+}0 = x} & -x = x \\
xy = yx & (xy)z = x(yz) & \\
x + y = y + x & (x + y) + z = x + (y + z) & \boldsymbol{x(y{+}z) = xy{+}xz}
\end{array}}$$

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 $\overline{x}$ 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*). It is known [22, 21] that circuits based on exclusive-or are smaller, on the average, than those using inclusive-or. A similar advantage should accrue symbolic representations. Like any other normal form, 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, of course, NP-hard [4]). Several possibilities are outlined in the next two sections. Section 4 proposes a Davis-Putnam-like method for satisfiability. We use a novel representation, called Bin-Lin, 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 Sect. 5) by some suggestions for practical improvements. Converting to the Bin-Lin representation is the subject of Sect. 6. Section 7 provides several relevant complexity results, including NP-completeness (via reduction from SAT, which 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. The latter results provide some justification as to why the proposed method may be efficient.

The main advantage of the proposed Bin-Lin representation is that it allows for a large measure of polynomial-time inference of new facts, using Gaussian elimination on the linear part and Horn methods for the binomial. This suggests incorporating more forward reasoning in a search-based satisfiability procedure than just Boolean constraint propagation. In Sect. 8, we apply the Boolean-ring representation in a framework—akin to Stålmarck's method [23] (see also [3]) and recursive learning [18]—that includes computing the intersection of sets of formulæ. Preliminary experimental results may be found in Sect. 9. This is followed by some brief comments.

## 2 Satisfiability

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 using pattern matching and repeatedly applying the emboldened ring axioms (on the previous page) from left to right to any subformula [14]. Tautologies reduce to 1; contradictions, to 0; contingent formulæ, to neither. But a normal form can of necessity be exponentially larger than the original. For example, the BNF of $(p + p')(q + q')(r + r')$ is $pqr + pqr' + pq'r + pq'r' + p'qr + p'qr' + p'q'r + p'q'r'$. Despite the fact that, in the process of normalization, terms that appear twice in a sum cancel each other out (since $x + x = 0$), this method is still, in reality, impractical.

An alternative [7, 15] to BNF 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 (permuting arguments, as necessary to match rule patterns), until no longer possible. 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 [16].

The Davis-Putnam-Logemann-Loveland (DPLL) procedure [8] was the first attempt to solve the satisfiability problem efficiently, 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 [15]. 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.

The traditional mechanical approach to satisfiability of a propositional formula is to assert 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 Sect. 8, we show how to 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. Equations being processed 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 distributivity).

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 DPLL and OBDD provers [6] include tautology, unit (Boolean constraint propagation), 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.

Performing simplification during preprocessing can also be useful. In [20], preprocessing CNF formulæ derived from circuit testing, bounded model checking, combinatorial equivalence checking, and superscalar processor verification, was 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 already mentioned, distributivity is the potentially expensive step (even when directed acyclic graphs are used for shared subterms). We propose a new representation, called Bin-Lin, that circumvents this problem.

A *linear equation* (over $\mathbb{Z}_2$) is a Boolean equation of 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, as in $x = y$ or $x + y = 0$, replace one by the other throughout $\mathcal{R}$.
6. *Simplification.* Inter-reduce one equation by another within $\mathcal{B}$ and within $\mathcal{L}$, by replacing occurrences in $\mathcal{B} \cup \mathcal{L}$ of the larger side (larger, in the given ordering $>$) of any equation $M = N$, say $M$, with the smaller side, $N$.
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 $\mathcal{R}$. Heuristics (e.g. [10]) can be applied to decide which variable $x$ to split on.

Equations in $\mathcal{B}$ and $\mathcal{L}$ are processed independently, except for the *Unit Rule* and *Equivalence Rule*. Splitting on $x$ and applying the rule for units eliminates $x$ from both $\mathcal{B} \cup \{x = 1\} \cup \mathcal{L}$ and $\mathcal{B} \cup \{x = 0\} \cup \mathcal{L}$.

This inference procedure is complete for propositional reasoning, because any Boolean formula can be converted into a Bin-Lin form preserving satisfiability, as will be explained in Sect. 6, and splitting with the *Unit Rule* suffices to test satisfiability of BR formulæ.

The *Simplification* step is not needed for completeness, but rather to improve search efficiency. All the steps, save *Splitting*, are encompassed by the replacement rules of Table 1. The choice of ordering $>$ is flexible: It may prefer long monomials, to keep equations short, or short ones, to maximize the likelihood of a simplifier applying.

One advantage of this mixed representation is that relatively fast methods exist for processing each of the two components; see Sect. 7. Software is readily 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 (non-unit) 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

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

**Table 1.** Inference rules for *replacing* antecendents with consequents ($A, B$ are any formulæ; $M, N$ are monomials, including variables and values; $x$ is any propositional variable; $c, d$ are either Boolean value; $A[N]$ is formula $A[M]$ with subformula $N$ instead of $M$; $\perp$ signifies unsatisfiability)

$$\frac{1=0}{\perp} \qquad \frac{\perp,\ A=B}{\perp} \qquad \frac{A=A}{}$$

$$\frac{MN=1}{M=1,\ N=1} \qquad \frac{M0=N}{N=0} \qquad \frac{M1=N}{M=N} \qquad \frac{Mxx=N}{Mx=N}$$

$$\frac{A+0=c}{A=c} \qquad \frac{A+1=1}{A=0} \qquad \frac{A+1=0}{A=1} \qquad \frac{A+x+x=c}{A=c} \qquad \frac{M+N=0}{M=N}$$

$$\frac{M=N,\ A[M]=B}{M=N,\ A[N]=B}\ M>N \qquad \frac{x+A=c,\ x+B=d}{x+A=c,\ A+B=c+d}\ x>B>A$$

To represent the clause $p \vee q$, a new variable $\widehat{p}$ for $\overline{p}$ is required. The clausal form of its negation, written as Boolean equations, is as follows:

$$\begin{array}{llll} (1)\ qs=q & (2)\ sp=s & (3)\ prt=pr & (4)\ rt=0 \\ & (5)\ p+\widehat{p}=1 & (6)\ q\widehat{p}=\widehat{p} & (7)\ pr=p\ . \end{array}$$

Equation (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 $\widehat{p} = 1$. The new (2) simplifies (1) to $q = 0$. Finally, $\widehat{p} = 1$ and $q = 0$ simplify (6) into the contradiction $1 = 0$, concluding the proof (*sans* splits). □

*Example 2.* The Pigeon-Hole Principle for 3 pigeons and 2 holes can be expressed as linear equations $\{a_1+a_2+1=0, b_1+b_2+1=0, c_1+c_2+1=0\}$ and binomial equations $\{x_i y_j = 0 \mid x_i, y_j \in A, x \neq y \vee i \neq j\}$, where $A = \{a_1, a_2, b_1, b_2, c_1, c_2\}$ and $x_i$ means that pigeon $x$ is in the $i$th hole. One split on $a_1$, with simplification, proves the principle. If $a_1 = 0$, then $a_2 = 1$ from the linear part, $b_2 = c_2 = 0$ from the binomial part, $b_1 = c_1 = 1$ from the linear part, leading to a contradiction $1 = 0$ from the binomial $b_1 c_1 = 0$. A similar contradiction obtains from $a_1 = 1$. □

## 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. A trivial case is when equations can belong to both sets. In fact, the Unit and Equivalence Rules of Sect. 4 belong to this case.

Since $\mathcal{B}$ is Horn, it can express cases where variables propagate values, such as those that can be obtained from equations in $\mathcal{L}$. For example, consider a linear equation $x + y + z = 1$. By assigning 1 to variables $x$ and $y$, the equation reduces to $z = 1$, since an even number of 1's cancel each other. Thus, we may conclude

**Table 2.** Transformation rules

| Gate | Bin-Lin |
|---|---|
| Wide-or: $F := \bigvee_i F_i$ | $\mu(F) = \prod_i \mu(F_i)$ |
| Wide-and: $F := \bigwedge_i F_i$ | $\nu(F) = \prod_i \nu(F_i)$ |
| Exclusive-or: $F := \bigoplus_i F_i$ | $\nu(F) = \sum_i \nu(F_i)$ |
| Negation: $F := \overline{G}$ | $\nu(F) = \mu(G)$ |
| Implication: $F := F_1 \Rightarrow F_2$ | $\mu(F) = \nu(F_1) \cdot \mu(F_2)$ |
| Equivalence: $F := F_1 \Leftrightarrow F_2 \Leftrightarrow \cdots \Leftrightarrow F_k$ | $\nu(F) = \sum_i \nu(F_i) + (k-1) \bmod 2$ |

that the binomial equation $xyz = xy$ (that is, $x$ and $y$ imply $z$) is a logical consequence of the linear equation. Furthermore, if we are lucky enough to have $xyz = 0$ (equivalent to $\overline{x} \vee \overline{y} \vee \overline{z}$) as a constraint, we can deduce $xy = 0$. In general, a linear equation $y_1 + y_2 + \cdots + y_k = 1$ implies the binomial $y_1 y_2 \cdots y_k = 0$, whenever $k$ is even, while $y_1 + y_2 + \cdots + y_k = 0$ implies $y_1 y_2 \cdots y_k = 0$ when $k$ is odd. The equation $y_1 + y_2 + \cdots + y_k = c$ breeds $k$ binomials of the form $y_1 y_2 \cdots y_k = y_1 \cdots y_{i-1} y_{i+1} \cdots y_k$ in the remaining two cases ($c = 1$ and $k$ odd, or $c = 0$ and $k$ even).

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 [12], 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 and 2 below).

Simplification within $\mathcal{B}$ tends to produce equations like $M = 0$, where $M$ is a monomial. However, substituting a variable $x$ in $M$ by its counterpart $1 + \widehat{x}$ can further expand it into a non-degenerate binomial equation.

## 6 Compilation

Any SAT formula encoded by a Boolean circuit can be recursively compiled into the Bin-Lin representation, as described in Table 2. Each logical gate $F$ is associated with two new variables, $\nu(F)$ and $\mu(F)$, together with the linear equation $\nu(F) + \mu(F) = 1$. Intuitively speaking, $\nu(F)$ is the structure variable for $F$, $\mu(F)$ is its negation, $\nu(x)$ is $x$, for input variable $x$, and $\mu(x)$ is a new variable $\widehat{x}$.

A SAT problem encoded in conjunctive normal form can also be compiled into the Bin-Lin representation in this way. A clause $l_1 \vee l_2 \vee \cdots \vee l_t$ can be easily converted into an equation by expanding $(l_1 + 1)(l_2 + 1) \cdots (l_t + 1) = 0$. If there are more than two positive literals amongst the $l_i$'s, the final equation will contain at least four monomials. To avoid this, we introduce new Boolean variables that serve as *complements* of the positive ones. When $l_i$ is a positive literal, a new variable $\widehat{l_i}$ and new equation $l_1 + \widehat{l_i} = 1$ are introduced, and $l_i + 1$ is replaced by $\widehat{l_i}$. Therefore, at most $2n$ variables remain after compilation.

Additional inference rules can help handle the positive variable $\nu(F)$ and the negative variable $\mu(F)$. Two variables $x$ and $y$ are called *complementary* if the linear equation $x + y = 1$ exists. We proceed as follows:

1. Replace any product of variables by 0 if it contains complementary variables.
2. Replace a negative variable $\widehat{x}$ by $x + 1$ in every linear equation other than $x + \widehat{x} = 1$ (and simplify).
3. Suppose we have $MM'x = N$ and $M'y = 0$, where $M, N, M'$ are monomials. Replace $MM'x = N$ by $MM' = N$ if $x$ and $y$ are complementary.
4. Replace $MN = 1$ by $M = 1$ and $N = 1$.
5. Suppose we have $MP = QR$ and $MQ = 0$. Replace $MP = QR$ by $MP = 0$ and $QR = 0$.
6. Remove an equation $x = M$ if $x$ is a variable that appears only in this equation.

*Example 3.* Consider, again, the instance in Example 1:

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

To prove this formula valid, we set it equal to false and try to find a counterexample. We compile it into the following:

$$\begin{array}{lll} (1)\ \widehat{t_1} = \widehat{p}\widehat{q} & (2)\ \widehat{t_2} = q\widehat{s} & (3)\ \widehat{t_3} = s\widehat{p} \\ (4)\ \widehat{t_4} = p\widehat{r} & (5)\ \widehat{t_5} = rp\widehat{t} & (6)\ t_6 = tr \\ (7)\ t_7 = t_1t_2t_3t_4t_5 & (8)\ \widehat{t_8} = t_7\widehat{t_6} & (9)\ t_8 = 0 \ . \end{array}$$

Note that the $t_i$ and the variables wearing hats are new. (We've omitted all the duality constraints.) It follows that (9) forces $t_7 = 1$ and $t_6 = 0$ in (8), which further forces $t_1 = t_2 = t_3 = t_4 = t_5 = 1$ in (7). Then repeatedly applying Rule 3 and the *Unit Rule*, we get $rp = 0$ in (5), $p = 0$ in (4), $s = 0$ in (3), $q = 1$ in (1), and finally $0 = 1$ for (2). □

In [23], all formulæ are represented as definitional "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.

## 7 Complexity

We next present several simple complexity results, including two polynomial-time subclasses (for $\mathcal{B}$ and $\mathcal{L}$). These results give some indication as to why our Bin-Lin method may be relatively effective.

Let Linear-BRSAT be the problem of solving a system of linear equations over the Boolean ring.

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

*Proof.* By Gaussian elimination (modulo 2). □

Note that, although Tseitin formulæ have an exponential lower bound for resolution [24, 2], they can be solved easily within $\mathcal{L}$ alone.

Let Binomial-BRSAT be the problem of solving a system of binomial equations.

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

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, which has linear complexity [13].

*Proof.* If $1 = 0$ is in $\mathcal{B}$, then it is unsatisfiable. Variables appearing in an equation $M = 1$ can each be assigned 1. These variables can then be used to simplify the other equations, and this process continues until all equations are of the form $M = 0$ or $M = N$. Observe that these remaining equations can be easily satisfied just by assigning 0 to all variables appearing in them. Implemented appropriately, the process terminates in linear time. □

Full-fledged computation of the Gröbner basis of $\mathcal{B}$ is not feasible, since its size can be exponential in the number of variables.[2]

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. We have already shown how to compile a set of clauses into the Bin-Lin representation (cf. Sect. 6). Thus:

**Theorem 3.** BRSAT *is NP-complete.*

It is easy to design a polynomial simplification algorithm that uses every binomial to rewrite every other binomial (Step 6), until nothing more can be reduced.[3]

## 8 Intersection Method

The core idea of recursive learning [18] is to learn as much as possible from shallow case splits, and to use the learned facts to prune the search space. Stålmarck's Prover [23] has used this innovative approach successfully for testing satisfiability of many large-scale industrial problems. Such saturate-by-intersection methods proceed via iterative deepening. Informally, the steps in this learning approach are as follows:

[2] We have not determined the precise complexity of computing an efficient representation of the basis.

[3] We have not, however, found a linear-time algorithm for this purpose.

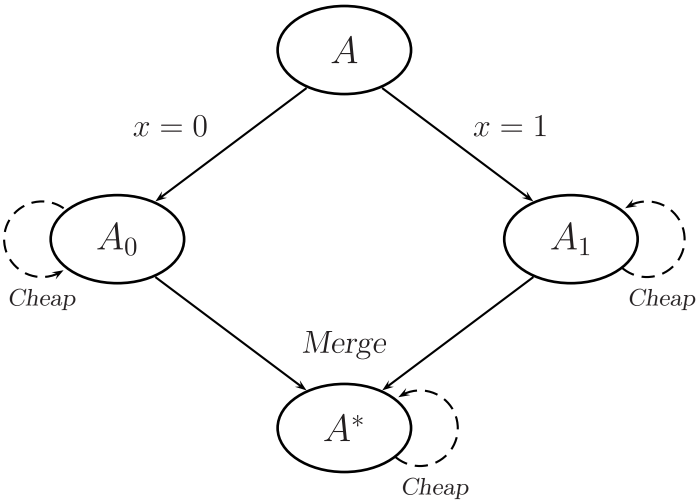

**Fig. 1.** Intersection-based learning method

1. Perform cheap (i.e. polynomial) inferences to reduce formulæ. 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 until maximum depth.

By "merging", we mean looking for consequences that hold in both cases. This, too, should be a polytime step. See Fig. 1.

Suppose, then, that we are provided with the following four polynomial procedures:

1. *Sat* $A$, which detects "obvious" cases of satisfiability of the formulæ in a set of formulæ $A$.
2. *Unsat* $A$, which detects "obvious" cases of unsatisfiability of $A$.
3. *Cheap* $A$, which performs simple, but incomplete, "reduction" inferences on $A$.
4. *Merge* $A, B$, which returns a set of formulæ that can be inferred from both $A$ and $B$.

Let *Th* $A$ be the theory (deductive closure) of $A$, $\bot$ denote unsatisfiability (falsehood, failure), and *Var* $A$ be the (unassigned) variables appearing in $A$. We need for the above procedures to satisfy the following requirements:

- *Sat* and *Unsat* are sound:

$$\mathit{Sat}\ A = T \Rightarrow (\mathit{Th}\ A)' \neq \emptyset$$
$$\mathit{Unsat}\ A = T \Rightarrow (\mathit{Th}\ A)' = \emptyset$$

  where $(\mathit{Th}\ A)'$ are the formulæ that are not theorems.

- *Sat* and *Unsat* do something:

$$\mathit{Var}\ A = \emptyset \Rightarrow (\mathit{Sat}\ A = T) \vee (\mathit{Unsat}\ A = T)$$

- *Cheap* is sound:

$$\mathit{Cheap}\ A \subseteq \mathit{Th}\ A$$

- *Cheap* is a closure operation:

$$\begin{aligned} A &\subseteq \mathit{Cheap}\ A \\ \mathit{Cheap}\ A &\subseteq \mathit{Cheap}\ (A \cup B) \\ \mathit{Cheap}\ (\mathit{Cheap}\ A) &= \mathit{Cheap}\ A \end{aligned}$$

- *Merge* is sound:

$$\begin{aligned} \mathit{Merge}\ A, B &\subseteq (\mathit{Th}\ A) \cap (\mathit{Th}\ B) \\ \mathit{Merge}\ A, \top = \mathit{Merge}\ \top, A &= \top \end{aligned}$$

- *Merge* does something:

$$\mathit{Merge}\ A, B \supseteq A \cap B$$

- Failure propagates:

$$\begin{aligned} \mathit{Sat}\ \bot &= F \\ \mathit{Unsat}\ \bot &= T \\ \mathit{Cheap}\ \bot &= \bot \\ \mathit{Merge}\ \bot, A = \mathit{Merge}\ A, \bot &\supseteq A \end{aligned}$$

With this framework, intersection-based learning (IBL) works as shown in the algorithm of Fig. 2. Assuming the above requirements, we have the following:

$$\begin{aligned} \mathit{IBL}(A) = \top &\Leftrightarrow (\mathit{Th}\ A)' \neq \emptyset \\ \mathit{IBL}(A) = \bot &\Leftrightarrow (\mathit{Th}\ A)' = \emptyset \\ \mathit{IBL}(A) \notin \{\top, \bot\} \Rightarrow \mathit{IBL}(A) &\subseteq \quad \mathit{Th}\ A \end{aligned}$$

The original Davis-Putnam satisfiability procedure [9] uses unit propagation and elimination for *Cheap*; it looks for the absence of complementary literals to declare *Sat*; it detects unsatisfiability in the form of an empty set; and merges by generating one round of resolvents.

For the Bin-Lin method, *Cheap* applies the simplifications described above; *Sat* includes checking that there is no 0 occurring in Bin and that all formulæ in Lin are of even parity; *Unsat* looks for $1 = 0$; *Merge* can be simple syntactic intersection. We have implemented such a scheme for the full Bin-Lin representation, as described in the next section.

Better yet, *Merge* could comprise limited generate-and-test. 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 $\mathcal{L}$.

```
$IBL(P) := I(P, Var\ P, N)$
where $I(B, V, n)$ is
    if $n = 0$ then return $Cheap\ B$
    $A := I(B, V, n - 1)$
    if $Unsat\ A$ then return $\bot$    $\|$    if $Sat\ A$ then return $\top$
    $V' := V \cap Var\ A$
    if $V' = \emptyset$ then return $A$
    choose $p :\in V$
    $V'' := V' \setminus p$
    $A_0 := I(A \cup \{\overline{p}\}, V'', n - 1)$    $\|$    $A_1 := I(A \cup \{p\}, V'', n - 1)$
    $A^* := Merge\ A_0, A_1$
    if $A^* \in \{\top, \bot\}$ then return $A^*$
    if $A = A^*$ then return $I(A, V'', n)$
              else return $I(A^*, Var\ A^*, n)$
```

**Fig. 2.** Generic intersection-based learning algorithm ($S \| S'$ means that execution of $S$ and $S'$ can be performed in parallel; $N$ is the maximum depth of splits for learning)

## 9 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 earlier, initial experiments with an implementation of a naïve Boolean-ring-based search method (of Sect. 2), the number of splits was reduced by 30% [17]. 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 conducted several sets of experiments to measure the usefulness of reduction techniques under the Bin-Lin framework:

**P:** This is a basic DPLL procedure, with learning of conflict clauses. Variables are split according to their given order. This set is intended as a baseline reference.

**I:** The intersection method described in Sect. 8, with syntactic intersection, is incorporated in this case. The maximum depth for intersection-based learning was set to 3.

**GI:** Gaussian elimination on linear equations is added to the implementation of Set **I**.

**HI:** Equivalence of variables are learned from binomial equations, together with Set **I**.

**GHI:** The features of Sets **GI** and **HI** are combined.

Over 700 satisfiable examples, taken from actual Intel Corp. hardware verification problems, were tested for each set of features. Not surprisingly, using combined simplification (**GHI**) performed best in terms of eliminating splits. Indeed, nearly 14% of these 700 examples have over 50% savings on splits with

**Table 3.** Representative runs.

| Input | | **P** | **I** | | **GI** | | **HI** | | **GHI** | | Over- |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Vars | Gates | Splits | Splits | Savings | Splits | Savings | Splits | Savings | Splits | Savings | head |
| 5 | 18 | 15 | 14 | 6.7% | 14 | 6.7% | 14 | 6.7% | 10 | 33.3% | 13 |
| 5 | 38 | 31 | 31 | 0.0% | 31 | 0.0% | 13 | 58.1% | 13 | 58.1% | 15 |
| 6 | 10 | 20 | 19 | 5.0% | 19 | 5.0% | 17 | 15.0% | 17 | 15.0% | 16 |
| 6 | 37 | 63 | 63 | 0.0% | 56 | 11.1% | 63 | 0.0% | 56 | 11.1% | 18 |
| 7 | 33 | 38 | 37 | 2.6% | 37 | 2.6% | 37 | 2.6% | 19 | 50.0% | 19 |
| 7 | 36 | 95 | 95 | 0.0% | 95 | 0.0% | 95 | 0.0% | 45 | 52.6% | 21 |
| 8 | 30 | 116 | 115 | 0.9% | 115 | 0.9% | 115 | 0.9% | 0 | 100% | 22 |
| 8 | 34 | 191 | 191 | 0.0% | 191 | 0.0% | 191 | 0.0% | 6 | 96.9% | 24 |
| 9 | 39 | 507 | 507 | 0.0% | 507 | 0.0% | 504 | 0.6% | 127 | 75.0% | 27 |
| 9 | 45 | 17 | 0 | 100% | 0 | 100% | 0 | 100% | 0 | 100% | 9 |
| 10 | 54 | 258 | 257 | 0.4% | 257 | 0.4% | 257 | 0.4% | 65 | 74.8% | 28 |
| 10 | 63 | 511 | 511 | 0.0% | 432 | 15.5% | 511 | 0.0% | 214 | 58.1% | 30 |
| 11 | 41 | 1794 | 1794 | 0.0% | 1794 | 0.0% | 896 | 50.1% | 896 | 50.1% | 33 |
| 11 | 46 | 1983 | 1983 | 0.0% | 1983 | 0.0% | 1983 | 0.0% | 991 | 50.0% | 33 |
| 11 | 52 | 1187 | 1187 | 0.0% | 593 | 50.0% | 1187 | 0.0% | 593 | 50.0% | 33 |

**GHI**. Some examples can be solved completely—without any splits—just with **I**; this phenomenon occurs more frequently for **GHI**.

Despite the heavy cost of merging, there was an overall average reduction of 3% in run time, compared to the baseline, since merging often produces simplifiers that greatly reduce the need for splitting.[4]

Table 3 shows some sample results. The savings for **GHI** are usually more than the sum of those obtained by **GI** and **HI**. This is due to the cross-fertilization between $\mathcal{B}$ and $\mathcal{L}$. The last column indicates the additional splits (on average) needed for applying the learning technique. In only two of the smallest cases does this overhead outweigh the benefit in terms of total splits.

## 10 Discussion

We have endeavored to show that using powerful simplification, such as provided by the Boolean-ring format, combined with the forward reasoning of intersection-based learning, can lead to a significant reduction in the number of variables on which splitting will need to be performed.

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

[4] The code wasn't designed to be competitive in terms of overall performance, but only as proof of concept. In particular, constant propagation over the Bin-Lin system was noticeably time consuming. Accordingly, there is no point in providing time comparisons with alternative approaches.

time, but simplification, though polynomial, is not linear. On the other hand, eliminating a split saves space, and may in the long run save time.

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 $\{\vee, \wedge, -\}$, and $\mathcal{G}$ over $\{1, +, \wedge\}$. As shown in [16], 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 formula, while Boolean algebra requires additional variables. This ability to preserve structure is also important for Stålmarck's method (cf. Sect. 8), 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 (see Example 2) 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.

## Acknowledgement

We are grateful to the referees for their comments.

## References

1. Baader, F., and Nipkow, T.: *Term Rewriting and All That.* Cambridge University Press (1998)
2. Ben-Sasson, E., and Wigderson, A.: Short proofs are narrow — resolution made simple. Journal of the ACM **48** (2001) 149–169
3. Björk, M.: A First Order Extension of Stålmarck's Method, Ph.D. thesis, Department of Computer Science and Engineering, Göteborg University, Sweden, 2006
4. 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
5. Bonacina, M. P. , and Dershowitz, N.: Abstract canonical inference. ACM Transactions on Computational Logic, to appear
6. Bryant, R. E.: Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys **24** (1992) 293–318
7. 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
8. Davis, M., Logemann, G., and Loveland, D.: A machine program for theorem proving. Communications of the ACM **5** (1962) 394–397
9. Davis, M., and Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7** (1960) 201–215

10. Dershowitz, N., Hanna, Z., and Nadel, A.: A clause-based heuristic for SAT solvers. Proc. Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT 2005; St. Andrews, Scotland), Bacchus, F., and Walsch, T., eds., Lecture Notes in Computer Science **3569**, Springer-Verlag, Berlin (June 2005) 46–60
11. Dershowitz, N., and Kirchner, C.: Abstract canonical presentations. Theoretical Computer Science **357** (2006) 53–69
12. 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
13. Dowling, W. F., and Gallier, J. H.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. Journal of Logic Programming **3** (1984) 267–284
14. 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
15. 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
16. Hsiang, J., and Huang, G. S.: Compact representation of Boolean formulas. Chinese Journal of Advanced Software Research **6**(2) 178–187 (1999)
17. 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)
18. Kunz, W., and Pradhan, D. K.: 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
19. McCune, W.: Solution of the Robbins problem. Journal of Automated Reasoning **19** (1997) 263-276
20. Marques-Silva, J. P.: Algebraic simplification techniques for propositional satisfiability. Technical Report RT/01/2000, INESC (2000)
21. Sasao, T.: And-exor expressions and their optimization. Logic Synthesis and Optimization, T. Sasao, ed., pp. 287–312, Kluwer, 1993
22. Sasao, T., and Besslich, P.: On the complexity of mod-2 sum PLA's. IEEE Transactions on Computers **C-39**(2) (Feb. 1990) 263–265
23. 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
24. Urquhart, A.: Hard examples for resolution. Journal of the ACM **34** (1987) 209–219