When Boolean Satisfiability Meets Gaussian Elimination in a Simplex Way

Cheng-Shen Han and Jie-Hong R. Jiang

Department of Electrical Engineering / Graduate Institute of Electronics Engineering
National Taiwan University, Taipei 10617, Taiwan
{hankf4@gmail.com, jhjiang@cc.ee.ntu.edu.tw}

Abstract. Recent research on Boolean satisfiability (SAT) reveals modern solvers' inability to handle formulae in the abundance of parity (XOR) constraints. Although XOR-handling in SAT solving has attracted much attention, challenges remain to completely deduce XOR-inferred implications and conflicts, to effectively reduce expensive overhead, and to directly generate compact interpolants. This paper integrates SAT solving tightly with Gaussian elimination in the style of Dantzig's simplex method. It yields a powerful tool overcoming these challenges. Experiments show promising performance improvements and efficient derivation of compact interpolants, which are otherwise unobtainable.

1 Introduction

For over a decade of intensive research, Boolean satisfiability (SAT) solving [2] on conjunctive normal form (CNF) formulae has become a mature technology enabling pervasive applications in hardware/software verification, electronic design automation, artificial intelligence, and other fields. The maturity on the other hand sharpens the boundary between what can and what cannot be achieved by the state-of-the-art solving techniques [23, 22, 10]. One clear limitation is their poor scalability in solving formulae that in part encode parity (XOR) constraints, which arise naturally in real-world applications such as cryptanalysis [21], model counting [11], decoder synthesis [14], arithmetic circuit verification, etc.

To overcome this limitation, there are prior attempts integrating special XOR handling into SAT solving [27, 4, 15, 6, 7, 25, 16, 26, 17]. Two different strategies have been explored. Non-interactive XOR handling, on the one hand, as pursued in [27, 6, 7] performs XOR reasoning and SAT solving in separate phases. Interactive XOR handling, on the other hand, as pursued in [4, 15, 25, 16, 26, 17] invokes XOR reasoning on-the-fly during SAT solving. Despite the expensiveness of XOR handling compared to CNF handling, positive results on conquering traditionally difficult problems have been demonstrated especially by the latter strategy, which is taken in this paper. Prior interactive methods can be further classified into two categories: inference-rule based [4, 15–17] and linear-algebra based [25, 26] XOR reasoning. The latter tends to be simpler in realization, and can be faster in performance as suggested by the empirical results in [17]. This paper adopts linear-algebra based computation [25, 26].

Regardless of the recent progress in XOR-reasoning, several challenges remain to be further addressed. Firstly, the deductive power of XOR-reasoning should be enhanced. To the best of the authors' knowledge, no current solver guarantees complete propagation/conflict detection for a given set of XOR-constraints with respect to some variable assignment. Secondly, the overhead of XOR-reasoning should be reduced, and the synergy between CNF solving and XOR-reasoning should be further strengthened. Thirdly, Craig interpolant generation is not supported by any current solver equipped with the XOR-reasoning capability. As interpolation becomes an indispensable tool for verification [19] and synthesis [13], compact interpolant derivation from combined CNF and XOR reasoning should be solicited.

The efforts of combining CNF and XOR reasoning share a common connection to Satisfiability Modulo Theories (SMT) [24]. There is, however, a subtle difference that makes these efforts distinct. The underlying CNF and XOR handlers encounter the same variables, whereas most, if not all, current SMT solvers with capability of producing Craig interpolants [8] assume the considered theories are of disjoint signatures. This difference makes recent advances in SMT solving and interpolation [20, 28, 5] not immediately helpful to alleviate the aforementioned challenges.

This paper tackles the above three challenges with the following results. Gauss-Jordan elimination (GJE) (in contrast to prior Gaussian elimination (GE) [25, 26]) is proposed for XOR-constraint processing in a matrix form. It admits complete detection of XOR-inferred propagations and conflicts. As the matrix is in the reduced row echelon form, the two-literal watching scheme fits in naturally for fast propagation/conflict detection, and for lazy and incremental matrix update in the style of Dantzig's simplex algorithm [9]. This simple data structure effectively reduces computation overhead and tightens the integration between CNF and XOR reasoning. Moreover, interpolant derivation rules are obtained for direct and compact interpolant generation. Experimental results suggest strong benefit of the proposed method in accelerating SAT solving. Promising improvements over the prior state-of-the-art solver [26] are observed. Moreover the results show efficient derivation of compact interpolants, which are otherwise unobtainable.

This paper is organized as follows. Preliminaries are given in Section 2. Section 3 presents our framework on SAT solving and XOR-reasoning; Section 4 covers interpolant generation in our framework. Experimental results and discussions are given in Section 5. Detailed comparison with the closest related work is performed in Section 6. Finally, Section 7 concludes this paper and outlines future work.

2 Preliminaries

We define terminology and notation to be used throughout this paper. Symbols $\wedge$, $\vee$, $\neg$, and $\oplus$ stand for Boolean AND, OR, NOT, and exclusive OR (XOR) operations, respectively. A *literal* is either a variable (i.e., in the positive phase) or the negation of a variable (i.e., in the negative phase). A (regular) *clause* is a disjunction of a set of literals. A Boolean formula is in *conjunctive normal form* (CNF) if it is expressed as a conjunction of a set of clauses. For a literal l, its corresponding variable is denoted as $var(l)$. Also since a clause is viewed as a set of literals, expression $l \in C$ denotes that l is a constituent literal of clause C, and $C' \subseteq C$ denotes C' is a subclause of C.

2.1 XOR Constraints

An XOR-*clause* is a series of XOR operations over a set of literals and/or Boolean constants $\{0, 1\}$. It equivalently represents a linear equation over GF(2), the Galois field of two elements. An XOR-clause is in the *standard form* if all of its literals appear in the positive phase. E.g., the XOR-clause $(x_1 \oplus \neg x_2 \oplus x_3)$ can be written in the standard form as $(x_1 \oplus x_2 \oplus x_3 \oplus 1)$, which equivalently represents the linear equation $x_1 \oplus x_2 \oplus x_3 = 0$. Note that an XOR-clause consisting of n variables translates into a conjunction of 2^{n-1} clauses with n literals each. E.g., the XOR-clause $(x_1 \oplus \neg x_2 \oplus x_3)$ can be clausified to the equivalent CNF $(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$. To avoid such exponential translation, an n-element XOR-clause $(l_1 \oplus \cdots \oplus l_n)$ can be divided into two XOR-clauses $(l_1 \oplus \cdots \oplus l_k \oplus y)$ and $(\neg y \oplus l_{k+1} \oplus \cdots \oplus l_n)$ by introducing a new fresh variable y. Some modern SAT solvers, e.g., CryptoMiniSat [26], can extract XOR-clauses from a set of regular clauses.[1]

A set of m XOR-clauses over n variables $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ can be considered as a system of m linear equations over n unknowns. Hence the XOR-constraints can be represented in a matrix form as $A\boldsymbol{x} = \boldsymbol{b}$, where A is an $m \times n$ matrix and $\boldsymbol{b}$ is an $m \times 1$ constant vector of values in $\{0, 1\}$. In the sequel, $A\boldsymbol{x} = \boldsymbol{b}$ is alternatively represented as a single Boolean matrix $M = [A|\boldsymbol{b}]$, where separation symbol "|" denotes matrix concatenation of A and $\boldsymbol{b}$, that is, matrix A is augmented with one more column $\boldsymbol{b}$.

Example 1. The three XOR-clauses c_1: $(x_1 \oplus \neg x_4)$, c_2: $(x_2 \oplus x_4)$, and c_3: $(x_1 \oplus \neg x_2 \oplus \neg x_3)$ correspond to the linear equations with the following matrix form.

$$\begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \end{array} \begin{array}{c} \begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & \boldsymbol{b} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

A matrix M can be reduced by Gaussian or Gauss-Jordan elimination to remove linearly dependent equations. Without loss of generality, we shall assume that matrix M has been preprocessed and is of full rank. In our treatment a matrix is often underdetermined, namely, there are more columns (unknowns) than rows (constraints). In the sequel, a matrix is also viewed as a set of rows.

This paper is concerned with the Boolean satisfiability of a formula given as a conjunction of regular clauses and/or XOR-clauses. Thus a formula is viewed

[1] Our implementation adopts the XOR extraction computation of CryptoMiniSat.

as a set of (XOR-)clauses. (In practice, the XOR-clauses can be given as part of the formula or deduced from the regular clauses.) In the sequel, a formula ϕ (respectively a system of linear equations $[A|\boldsymbol{b}]$) over variables $\boldsymbol{x}$ subject to some truth assignment $\alpha : \boldsymbol{x'} \to \{0,1\}$ on variables $\boldsymbol{x'} \subseteq \boldsymbol{x}$ is denoted as $\phi|_\alpha$ (respectively $[A|\boldsymbol{b}]|_\alpha$). That is, $\phi|_\alpha$ (respectively $[A|\boldsymbol{b}]|_\alpha$) is the induced formula of ϕ (respectively linear equations $[A|\boldsymbol{b}]$) with variable x_i substituted with its truth value $\alpha(x_i)$. We represent α with a characteristic function. E.g., $\alpha = \neg x_1 x_2 \neg x_3$ denotes $\alpha(x_1) = 0$, $\alpha(x_2) = 1$, and $\alpha(x_3) = 0$.

2.2 Resolution Refutation and Craig Interpolation

Assume literal l is in clause C_1 and $\neg l$ in C_2. A *resolution* of clauses C_1 and C_2 on $var(l)$ yields a new clause C containing all literals in C_1 and C_2 except for l and $\neg l$. The clause C is called the *resolvent* of C_1 and C_2. For an unsatisfiable CNF formula, there always exists a resolution sequence, referred to as a *resolution refutation*, leading to an empty-clause resolvent. Resolution refutation has a tight connection to Craig interpolants.

Theorem 1 (Craig Interpolation Theorem). *[8]*
For two Boolean formulae ϕ_A and ϕ_B with $\phi_A \wedge \phi_B$ unsatisfiable, there exists a Boolean formula I_A referring only to the common variables of ϕ_A and ϕ_B such that $\phi_A \to I_A$ and $I_A \wedge \phi_B$ is unsatisfiable.

The Boolean formula I_A is referred to as the *interpolant* of ϕ_A with respect to ϕ_B. When ϕ_A and ϕ_B are in CNF, a refutation proof of $\phi_A \wedge \phi_B$ is derivable from a SAT solver such as MiniSat [10]. Further, an interpolant circuit I_A can be constructed from the refutation proof in linear time [20].

3 Satisfiability Solving under XOR Constraints

Modern SAT solvers are based on the conflict-driven clause learning (CDCL) mechanism. Our proposed decision procedure is built on top of the modern solvers. Figure 1 sketches the pseudo code, where lines 2 and 13-16 are inserted for special XOR-handling. In line 2, XOR-clauses are extracted from the input formula ϕ. Let $A\boldsymbol{x} = \boldsymbol{b}$ be a system of linearly independent equations derived from these XOR-clauses. Then $M = [A|\boldsymbol{b}]$. If M is empty, lines 13-16 take no effect and the pseudo code works the same as the standard CDCL procedure. On the other hand, when M contains a non-empty set of linear equations, the procedure *Xorplex* in line 13 deduces implications or conflicts whenever they exist from M with respect to a given variable assignment α. In the process, matrix M may be changed along the computation. When implication (or propagation) happens, α is expanded to include newly implied variables. If any implication or conflict results from *Xorplex*, in line 15 essential information is added to ϕ in the form of learnt clauses, which not only reduces search space but also facilitates future conflict analysis.

```
SatSolve
  input: Boolean formula φ
  output: SAT or UNSAT
  begin
     α := ∅;
     M := ObtainXorMatrix(φ);
     repeat
        (status, α) := PropagateUnitImplication(φ, α);
        if status = conflict
           if conflict at top decision level
              return UNSAT;
           φ := AnalyzeConflict&AddLearntClause(φ, α);
           α := Backtrack(φ, α);
        else
           if all variables assigned
              return SAT;
           (status, α) := Xorplex(M, α);
           if status = propagation or conflict
              φ := AddXorImplicationConflictClause(φ, M, α);
              continue;
           α := Decide(φ, α);
  end
```

Fig. 1. Algorithm: SatSolve.

3.1 XOR Reasoning

Before elaborating our XOR-reasoning technique, we show an example motivating the adoption of Gauss-Jordan elimination.

Example 2. Consider the following matrix triangularized by Gaussian elimination.

$$[A|\boldsymbol{b}] = \begin{pmatrix} 1\,1\,1\,1\,1\,0 \\ 0\,1\,1\,1\,1\,0 \\ 0\,0\,1\,1\,1\,1 \\ 0\,0\,0\,1\,1\,1 \end{pmatrix}$$

No implication can be deduced from it. With Gauss-Jordan elimination, however, it is reduced to the following diagonal matrix.

$$[A'|\boldsymbol{b}'] = \begin{pmatrix} 1\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,1 \\ 0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,1\,1\,1 \end{pmatrix}$$

The values of the first three variables can be determined from the four equations. Therefore Gaussian elimination (as is used by CRYPTOMINISAT) is strictly weaker than Gauss-Jordan elimination in detecting implications and conflicts.

The efficacy of XOR-handling in the pseudo code of Figure 1 is mainly determined by the procedure *Xorplex*. In essence, two factors, deductive power and computational efficiency, need to be considered in realizing *Xorplex*. We show how the two-literal watching scheme in unit propagation [22] fits incremental Gauss-Jordan elimination in a way similar to the simplex method to support lazy update. Consequently, *Xorplex* can be implemented efficiently and has complete power deducing implications and conflicts whenever they exist.

In the simplex method, the variables of the linear equations $A\boldsymbol{x} = \boldsymbol{b}$ are partitioned into m *basic variables* and $(n - m)$ *nonbasic variables* assuming that the $m \times (n + 1)$ matrix $[A|\boldsymbol{b}]$ is of full rank and $m < n$. Matrix $[A|\boldsymbol{b}]$ is diagonalized to $[I|A'|\boldsymbol{b}']$, where I is an $m \times m$ identity matrix and A' is an $m \times (n - m)$ matrix, by Gauss-Jordan elimination such that the m basic and $(n - m)$ nonbasic variables correspond to the columns of I and A', respectively. Note that diagonalizing $[A|\boldsymbol{b}]$ to $[I|A'|\boldsymbol{b}']$ may incur column permutation, which is purely for the ease of visualization to make the columns indexed by the basic variables adjacent to form the identity matrix. In practice, such permutation is unnecessary and not performed. By the simplex method, a basic variable and a nonbasic variable may be interchanged in the process of searching for a feasible solution optimal with respect to some linear objective function. The basic variable to become nonbasic is called the *leaving variable*, and the nonbasic variable to become basic is called the *entering variable*. Although the simplex method was proposed for linear optimization over the reals, the matrix operation mechanism works for our considered XOR-constraints, i.e., linear equations over GF(2).

The problem of XOR-constraint solving is formulated as follows. Given a system of linear equations $A\boldsymbol{x} = \boldsymbol{b}$ and a partial truth assignment α to variables $\boldsymbol{x}' \subseteq \boldsymbol{x}$, if the induced linear equations $[A|\boldsymbol{b}]|_\alpha$ with respect to α are consistent, derive *all* implications to the non-assigned variables $\boldsymbol{x}\backslash\boldsymbol{x}'$. Otherwise, detect a conflicting assignment to $\boldsymbol{x}'$ that leads to the inconsistency. In fact, Gauss-Jordan elimination achieves this goal as the following proposition asserts.

Proposition 1. *Given a set of* XOR-*constraints* $A\boldsymbol{x} = \boldsymbol{b}$ *and a partial truth assignment* $\alpha : \boldsymbol{x}' \rightarrow \{0, 1\}$ *for* $\boldsymbol{x}' \subseteq \boldsymbol{x}$*, Gauss-Jordan elimination on the induced linear equations* $[A|\boldsymbol{b}]|_\alpha$ *detects all implications to the non-assigned variables* $\boldsymbol{x}\backslash\boldsymbol{x}'$ *if* $[A|\boldsymbol{b}]|_\alpha$ *is consistent, or detects a conflict if* $[A|\boldsymbol{b}]|_\alpha$ *is inconsistent.*

Proof. The proposition follows from the soundness and completeness of GJE for solving a system of linear equations. ■

To equip complete power in deducing implications and conflicts, procedure *Xorplex* of Figure 1 maintains $M|_\alpha$ in a reduced row echelon form. Since *Xorplex* is repeatedly applied under various assignments α during SAT solving, Gauss-Jordan elimination needs to be made fast. A two-literal[2] watching scheme is proposed to make incremental updates on M in a lazy fashion, thus avoiding

[2] Since the variables in M are of positive phases, there is no need to distinguish "two-literal" or "two-variable" watch.

wasteful computation. Essentially, the following invariant is maintained for M at all times.

Invariant: Given a partial truth assignment α to the variables of matrix $M = [A|\boldsymbol{b}]$, for each row r of M two non-assigned variables are watched. Particularly, the first watched variable (denoted $w_1(r)$) must be a basic variable and the second watched variable (denoted $w_2(r)$) must be a nonbasic variable.

Note that, by this invariant, we assume each row of A contains at least three 1-entries. The reason is that a row without any 1-entry corresponds to either a tautological or conflicting equation, a row with one 1-entry corresponds to an immediate implication, and a row with two 1-entries asserts the equivalence or complementary relation between two variables and is handled specially. Note also that the number of 1-entries in some row of A can possibly be reduced to two later due to incremental Gauss-Jordan elimination. In this situation this row is removed from M and handled specially.

To maintain the invariant, when the two watched variables of some row in M are non-assigned, no action needs to be taken on this row for Gauss-Jordan elimination. On the other hand, actions need to be taken for the following two cases. For the first case, when variable $w_2(r)$ is assigned, another non-assigned nonbasic variable in row r is selected as the new second watched variable. No other rows are affected by this action. For the second case, when $w_1(r)$ is assigned and thus becomes the leaving variable, a non-assigned nonbasic variable in row r needs to be selected as the entering variable. The column c of the entering variable then undergoes the *pivot operation*, which performs row operations (additions) forcing all entries of c to be 0 except for the only 1-entry appearing at row r. Note that the pivot operation may possibly cause the vanishing of variable $w_2(r')$ from another row r'. In this circumstance a new non-assigned nonbasic variable needs to be selected for the second watched variable in row r', that is, the first case. Note that the process of maintaining the invariant always terminates because for every row r the update of $w_1(r)$ can occur at most once, and thus a row is visited at most m times for M of m rows.

When the invariant can no longer be maintained on some row r of M under α, either of the following two cases happens. Firstly, all variables of r are assigned. In this case the linear equation of r is either satisfied or unsatisfied. For the former, no further action needs to be applied on r; for the latter, *Xorplex* returns the detected conflict. Secondly, only variable $w_1(r)$ (respectively variable $w_2(r)$) is non-assigned. In this case, the value of $w_1(r)$ (respectively $w_2(r)$) is implied. Accordingly, α is expanded with $w_1(r)$ (respectively $w_2(r)$) assigned to its implied value.

Upon termination, procedure *Xorplex* leads to one of the four results: 1) propagation, 2) conflict, 3) satisfaction, and 4) indetermination. Only the first two cases yield useful information for CDCL SAT solving. The information is provided by procedure *AddXorImplicationConflictClause* in line 15 of the pseudo code in Figure 1. In the propagation case, the corresponding rows in M that implications occur are converted to learnt clauses. In the conflict case, the conflicting row in M is converted to a learnt clause. For example, a propagation (respectively conflict) occurs at a row corresponding to the linear equation $x_1 \oplus x_2 \oplus x_3 = 0$ under $\alpha(x_1) = 0, \alpha(x_2) = 1$ (respectively $\alpha(x_1) = 0, \alpha(x_2) = 1, \alpha(x_3) = 0$). Then the learnt clause $(x_1 \vee \neg x_2 \vee x_3)$ is produced.

3.2 Implementation Issues

In our actual realization, an $m \times (n+1)$ matrix M is implemented with a one-dimensional bit array, similar to [26]. Thereby matrix row addition is performed by bitwise XOR operation; a row addition operation translates to n/k bitwise XOR operations, where k is the bit width of a computer word. Moreover, similar to [26], if two XOR-constraint sets have disjoint support variables, they are represented by two individual matrices rather than a single matrix for the sake of memory and computational efficiency.

To support two-literal (or two-variable) watch on M, a watch list is maintained, which provides fast lookup for which rows of M to update when a variable is assigned. To maintain the invariant of two-literal watching, the most costly computation occurs when the basic variable of some row is assigned. It may incur in the worst case $O(m)$ row additions to set a new basic variable for the row. Nevertheless notice that this action cannot make the basic variables of other rows be assigned, and therefore no chain reaction is triggered. For an entire Gauss-Jordan elimination, the time complexity is $O(m^2 n)$.

4 Refutation and Interpolation

This section shows how Craig interpolants can be compactly constructed under the framework of *SatSolve*, which combines CDCL-based clause reasoning and GJE-based XOR-constraint solving. Although interpolants for combined propositional and linear arithmetic theories are available under the framework of SMT [20, 28, 5], they are not directly applicable in our context due to the underlying assumption of most SMT solvers that requires the considered theories to be of disjoint signatures. On the other hand, although theoretically XOR-constraints can always be expressed in CNF and thus propositional interpolation is sufficient, practically such CNF formulae are hard to solve and even if solvable their interpolants can be unreasonably large. A new method awaits to be uncovered.

4.1 Interpolant Generation

For problem formulation, consider interpolant generation for a given unsatisfiable formula $\phi = \phi_A \wedge \phi_B$ with the set V_A of variables of ϕ_A, V_B of ϕ_B, and V_{AB} of common variables shared by ϕ_A and ϕ_B. Let $\phi_A = \varphi_A \wedge \psi_A$ and $\phi_B = \varphi_B \wedge \psi_B$, where φ_A and φ_B are CNF formulae and ψ_A and ψ_B are XOR-constraints. Let M_A (respectively M_B) be the matrix form of the set of linear equations expressed by ψ_A (respectively ψ_B). Then the union of the rows of M_A and M_B corresponds to the matrix M denoted in the previous section.[3]

[3] For an XOR-constraint whose constituent clauses are not all implied by ϕ_A or by ϕ_B, it is not included in M when interpolant derivation is concerned.

If $\varphi_A \wedge \varphi_B$ is already unsatisfiable, the following *clause interpolation rules* [20] suffice to produce the interpolant.

$$\textsc{Cls-A}\ \frac{}{C\colon \langle\ \bigvee_{l \in C, var(l) \in V_{AB}} l\ \rangle}\ C \in \varphi_A$$

$$\textsc{Cls-B}\ \frac{}{C\colon \langle\ 1\ \rangle}\ C \in \varphi_B$$

$$\textsc{Cls-ResA}\ \frac{C_1 \vee l\colon \langle\ I_1\ \rangle \qquad C_2 \vee \neg l\colon \langle\ I_2\ \rangle}{C_1 \vee C_2\colon \langle\ I_1 \vee I_2\ \rangle}\ var(l) \in V_A \backslash V_{AB}$$

$$\textsc{Cls-ResB}\ \frac{C_1 \vee l\colon \langle\ I_1\ \rangle \qquad C_2 \vee \neg l\colon \langle\ I_2\ \rangle}{C_1 \vee C_2\colon \langle\ I_1 \wedge I_2\ \rangle}\ var(l) \in V_B$$

Similarly, if $\psi_A \wedge \psi_B$ is already unsatisfiable, the *inequality interpolation rules* [20] suffice for interpolant derivation. They are modified in our context for linear equations over GF(2) in the following.

$$\textsc{Xor-A}\ \frac{}{[\boldsymbol{a}^T|b]\colon \langle\ [\boldsymbol{a}^T|b]\ \rangle}\ [\boldsymbol{a}^T|b] \in M_A$$

$$\textsc{Xor-B}\ \frac{}{[\boldsymbol{a}^T|b]\colon \langle\ [\boldsymbol{0}^T|0]\ \rangle}\ [\boldsymbol{a}^T|b] \in M_B$$

$$\textsc{Xor-Sum}\ \frac{[\boldsymbol{a_1}^T|b_1]\colon \langle\ [\boldsymbol{a_1^*}^T|b_1^*]\ \rangle \qquad [\boldsymbol{a_2}^T|b_2]\colon \langle\ [\boldsymbol{a_2^*}^T|b_2^*]\ \rangle}{[\boldsymbol{a_1}^T|b_1] + [\boldsymbol{a_2}^T|b_2]\colon \langle\ [\boldsymbol{a_1^*}^T|b_1^*] + [\boldsymbol{a_2^*}^T|b_2^*]\ \rangle}$$

In the above rules, a partial interpolant, shown in the angle brackets, is associated to each linear equation. Superscript "T" and operator "+" denote matrix transpose and (modulo 2) matrix addition, respectively. The correctness of these derivation rules is immediate from prior results [20].

Complication arises, however, in interpolant generation when the refutation proof of ϕ involves both clausal resolution and XOR linear arithmetic. Essentially the partial interpolant of a constituent clause of a linear equation is needed. Let C be a constituent clause of equation $\boldsymbol{a}^T\boldsymbol{x} = b$, whose partial interpolant is ${\boldsymbol{a}^*}^T\boldsymbol{x} = b^*$. Then the following derivation rule applies.

$$\textsc{XorToCls}\ \frac{}{C\colon \langle\ C^* \vee ({\boldsymbol{a}^*}^T\boldsymbol{x} = b^*)|_{\neg C}\ \rangle}\ C \in [\boldsymbol{a}^T|b]\colon \langle\ [{\boldsymbol{a}^*}^T|b^*]\ \rangle$$

where $C^* \subseteq C$ with $C^* = \{l \in C \mid var(l) \in V_{AB} \cap Var({\boldsymbol{a}^*}^T\boldsymbol{x} = b^*)\}$ for $Var(E)$ denoting the variable set involved in equation E.

Example 3. Consider two equations $[\underline{1}\ 0\ \underline{1}\ 1\ 1\ 1] \in M_A$ and $[0\ \underline{1}\ 0\ \underline{1}\ 1\ 1] \in M_B$ in matrix form over variables $\{x_1, \ldots, x_5\}$ with $V_{AB} = \{x_3, x_4, x_5\}$, where the underlined variables are watched. Assume x_1 and x_2 are the basic variables. Under assignment $(x_1 = 0, x_2 = 1)$, the first and second equations are updated to $[1\ 0\ \underline{1}\ \underline{1}\ 1\ 1]$ and $[\underline{1}\ \underline{1}\ 1\ 0\ 0\ 0]$, respectively, with new basic variables x_2 and x_4. The partial interpolant of $[1\ 1\ 1\ 0\ 0\ 0]$, i.e., equation $x_1 \oplus x_2 \oplus x_3 = 0$, is derived as follows.

$$\dfrac{\dfrac{}{[1\ 0\ 1\ 1\ 1\ 1]\colon \langle\ [1\ 0\ 1\ 1\ 1\ 1]\ \rangle} \qquad \dfrac{}{[0\ 1\ 0\ 1\ 1\ 1]\colon \langle\ [0\ 0\ 0\ 0\ 0\ 0]\ \rangle}}{[1\ 1\ 1\ 0\ 0\ 0]\colon \langle\ [1\ 0\ 1\ 1\ 1\ 1]\ \rangle}$$

Since implication occurs with $x_3 = 1$, a learnt clause $(x_1 \vee \neg x_2 \vee x_3)$ is generated, which is a constituent clause of the clause set $\{(\neg x_1 \vee \neg x_2 \vee \neg x_3), (\neg x_1 \vee x_2 \vee x_3), (x_1 \vee \neg x_2 \vee x_3), (x_1 \vee x_2 \vee \neg x_3)\}$ defined by $x_1 \oplus x_2 \oplus x_3 = 0$. By rule XorToCls, the partial interpolant of the learnt clause equals

$$\begin{aligned} & x_3 \vee (x_1 \oplus x_3 \oplus x_4 \oplus x_5 = 1)|_{\neg x_1 x_2 \neg x_3} \\ = \ & x_3 \vee (x_4 \oplus x_5). \end{aligned}$$

Note that any clause implied by ϕ_A (respectively ϕ_B) can be considered as a clause of ϕ_A (respectively ϕ_B). Similarly any linear equation derivable from M_A (respectively M_B) can be viewed as a linear equation of M_A (respectively M_B). With this observation, one can verify that the partial interpolant derivation for a constituent clause of a linear equation in M_A (respectively M_B) reduces to McMillan's clause interpolation rule for clauses of ϕ_A (respectively ϕ_B). The general correctness of rule XorToCls is asserted by the following proposition.

Proposition 2. *The partial interpolant derived from rule* XorToCls *for* $C \in [\boldsymbol{a}^T|b]$ *is consistent with that derived from the clause interpolantion rules applied on the clauses clausified from* XOR*-constraints.*

Proof. Observe that every linear equation $E = [\boldsymbol{a}^T|b]$ derivable from M can always be expressed as a summation of two equations, one, E_A, derived from a linear combination of equations in M_A and the other, E_B, from M_B. (In fact E_A is the partial interpolant of E.) For C be a constituent clause of E, we show that its partial interpolant derived by XorToCls is the same as that derived by the clause interpolation rules applied on the resolution sequence leading to C from the clauses of E_A and E_B.

Let the variables appearing in E_A and E_B be divided into five disjoint (possibly empty) subsets: V_1 for those in E_A but not in E_B and V_{AB}, V_2 for those in E_A and V_{AB} but not in E_B, V_3 for those in both E_A and E_B (surely in V_{AB}), V_4 those in E_B and V_{AB} but not in E_A, and V_5 for those in E_B but not in E_A and V_{AB}. Then the variable set of C must be $V_1 \cup V_2 \cup V_4 \cup V_5$.

Because the system consisting of two linear equations $E_A|_{\neg C}$ and $E_B|_{\neg C}$ must be unsatisfiable (due to the fact C being a clause of the summation of E_A and E_B), by the completeness of resolution, C can be derived by resolution on variables V_3 from the clauses of E_A and E_B, more precisely, those clauses whose literals are consistent with C. Since the clauses of E_A and E_B can be considered as clauses in ϕ_A and ϕ_B, respectively, by rule Cls-A the partial interpolants for E_A clauses are subclauses with V_1 variables being removed, and by rule Cls-B the partial interpolants for E_B clauses equal constant 1. Since only V_3 variables are resolved, the partial interpolants are built from pure conjunction operation. As can be verified, the so-derived partial interpolant of C is the same as that of XorToCls regardless of the detailed resolution steps. ■

In essence rule XorToCls provides a short cut in generating interpolants. The xor-constraint reasoning circumvents unnecessary complex fine-grained resolutions and, perhaps more importantly, enforces its equivalent clausal resolution steps being performed within ϕ_A and ϕ_B locally whenever possible. These advantages make compact interpolants derivable from simple generation rules.

4.2 Implementation Issues

For an xor-equation derived as a summation of rows $R \subseteq M$, its partial interpolant is simply the summation of rows $R \cap M_A$. To derive the partial interpolant, the $m \times (n+1)$ matrix M is augmented to $M^* = [M|M_A^*]$ by concatenating M with another $m \times n$ matrix M_A^*, which is derived from M by removing the last column and replacing every row belonging to M_B with a row of 0's. Essentially the sub-matrix M_A^* of M^* maintains the partial interpolants at any moment of Gauss-Jordan elimination on M^*. More precisely, in $[M|M_A^*]$, a row in the sub-matrix M_A^* corresponds to the partial interpolant of the same row in the sub-matrix M.

5 Experimental Results

The proposed SAT solving method, named SimpSat, was implemented in the C^{++} language based on CryptoMiniSat 2.9.1 (CMS) [26], a state-of-the-art solver equipped with Gaussian elimination. All experiments were conducted on a Linux workstation with a 3.3 GHz Intel Xeon CPU and 64 GB memory. Benchmark examples with many xor-constraints were taken for experiments.

The first experiment compares our method with CMS on cryptanalysis benchmarks [26]. Four ciphers, Bivium, Trivium, HiTag-2, and Grain, were included with 100 instances each. For fair comparison, same parameters were applied on CMS and SimpSat. The results are shown in Table 1, where three methods were applied, namely, CMS$^-$ (CMS with GE disabled), CMS$^+$ (CMS with GE enabled), and SimpSat. The total CPU time averaging over the 100 instances is reported in the second, third, and seventh columns; the portion spent on GE is reported in the fourth and eighth columns; the number of invoked GE calls averaging over the 100 instances is shown in the fifth and ninth columns; the utility of GE, defined as the ratio of the number of useful GE calls (where implication or conflict happened) to that of all GE calls, is listed in the sixth and tenth columns; the speedup of SimpSat over CMS$^+$ in terms of the average

Table 1. Performance Comparison on Cryptanalysis Benchmarks.

Instance	CMS⁻	CMS⁺				SimpSat					
	Time (sec)	Time (sec)	Time GE (sec)	#GE	GE Util (%)	Time (sec)	Time GE (sec)	#GE	GE Util (%)	Spdup over CMS⁺	GE Spdup per call
`Bivium-45`	58.39	65.59	14.70	392200.37	38.39	30.65	9.86	545247.52	63.51	2.14	2.07
`Bivium-46`	29.47	26.75	8.09	214578.71	40.99	17.28	5.89	317329.83	64.09	1.55	2.03
`Bivium-47`	18.80	17.99	5.79	157721.70	42.39	10.53	4.01	216342.67	65.52	1.71	1.97
`Bivium-48`	12.50	11.48	3.91	109732.89	43.74	7.43	2.85	151763.17	66.12	1.55	1.90
`Bivium-49`	6.51	6.40	2.70	77970.24	46.91	3.55	1.51	80411.71	66.83	1.80	1.85
`Bivium-50`	5.89	4.76	1.97	59077.97	47.25	2.51	1.23	61643.62	67.55	1.90	1.68
`Bivium-51`	2.79	2.43	1.13	36940.35	48.48	1.32	0.65	34248.57	67.64	1.84	1.59
`Bivium-52`	1.15	1.31	0.66	23139.51	49.88	0.77	0.36	18385.35	68.02	1.71	1.46
`Bivium-53`	0.73	0.72	0.40	17602.80	52.45	0.44	0.22	10868.63	69.03	1.63	1.14
`Bivium-54`	0.59	0.58	0.27	11318.58	50.98	0.38	0.13	7248.02	68.99	1.51	1.29
`Bivium-55`	0.42	0.40	0.22	10905.49	53.43	0.26	0.11	5540.11	69.54	1.52	0.98
`Bivium-56`	0.24	0.23	0.12	5958.18	54.16	0.16	0.06	2842.08	71.29	1.40	0.94
`Trivium-151`	264.88	2314.04	60.81	1221568.44	36.19	131.14	32.21	1721026.43	60.69	1.76	2.66
`Trivium-152`	156.83	140.32	39.95	801775.05	38.31	70.59	19.88	1100015.00	61.63	1.99	2.76
`Trivium-153`	72.97	64.18	22.36	437299.75	41.06	30.76	9.91	581075.51	63.13	2.09	3.00
`Trivium-154`	57.76	45.57	16.20	316464.91	42.57	20.48	6.50	408162.70	63.38	2.23	3.21
`Trivium-155`	31.68	25.90	9.57	190731.45	42.65	13.38	4.57	268314.49	63.09	1.93	2.94
`Trivium-156`	15.39	16.72	6.56	133200.84	44.45	8.47	3.06	186959.82	64.14	1.97	3.01
`Trivium-157`	15.15	14.56	5.85	124892.01	45.36	7.14	2.63	164411.23	64.66	2.04	2.93
`HiTag2-9`	313.58	308.30	2.29	355291.70	7.39	235.89	5.50	1436229.45	22.34	1.31	1.69
`HiTag2-10`	146.93	143.32	1.40	208920.52	7.40	115.45	3.18	860728.62	22.13	1.24	1.81
`HiTag2-11`	60.87	61.02	0.71	104612.13	7.20	49.63	1.56	425575.32	21.85	1.23	1.86
`HiTag2-12`	27.50	27.03	0.40	57723.48	7.49	23.17	0.84	230317.78	21.21	1.17	1.90
`HiTag2-13`	14.02	13.63	0.26	38584.40	7.21	11.64	0.48	131037.02	21.31	1.17	1.82
`HiTag2-14`	6.24	6.27	0.13	17048.46	6.94	5.37	0.26	68325.91	20.73	1.17	2.02
`HiTag2-15`	2.93	2.90	0.07	10649.43	5.77	2.52	0.15	37043.91	20.55	1.15	1.72
`Grain-106`	688.50	712.27	35.23	841125.77	8.62	690.27	57.17	3347468.01	30.00	1.03	2.45
`Grain-107`	269.72	242.70	17.15	373763.02	8.48	211.73	24.24	1429181.91	29.79	1.15	2.71
`Grain-108`	1114.20	119.86	11.41	227777.96	9.33	112.99	14.32	872262.35	31.50	1.06	3.05
`Grain-109`	68.83	85.55	8.80	171188.65	9.87	70.54	9.63	592547.87	32.43	1.21	3.16

total CPU time (the ratio of that spent by CMS⁺ to that spent by SimpSat) is displayed in the eleventh column; the speedup of SimpSat over CMS⁺ in terms of the average CPU time taken per GE call (the ratio of that spent by CMS⁺ to that spent by SimpSat) is calculated in the last column. To summarize, SimpSat exhibited stronger deductive power (as seen by comparing the sixth and tenth columns) in shorter computation time (as seen from the last column) compared with CMS⁺. Thereby SimpSat achieved average speedup of 1.69x, 2.00x, 1.21x, and 1.11x for Bivium, Trivium, HiTag-2, and Grain, respectively. Figure 2 compares the performance of SimpSat and CMS⁺ on all of the cryptanalysis benchmarks. The CPU times spent by SimpSat and CMS⁺ are shown on the y-axis and x-axis, respectively. As can be seen, SimpSat steadily outperformed CMS⁺.

Under a similar setting, experiments were performed on equivalence checking benchmarks for Altera CRC (cyclic redundancy check) circuits [1].[4] Table 2 compares the performances of ABC cec command [3], CMS⁻, CMS⁺, and Simp-

[4] A benchmark was prepared by creating a miter structure comparing a design against its synthesized version using a script of ABC commands `dc2`, `dch`, `balance -x`.

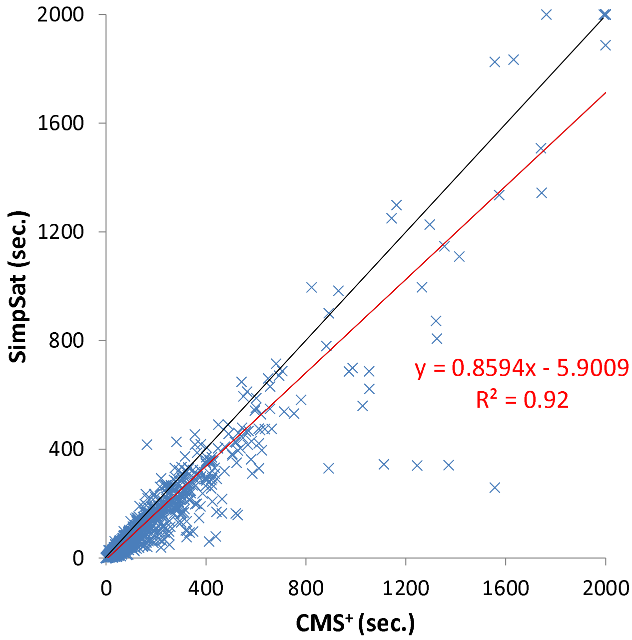

Fig. 2. Rumtimes on cryptanalysis benchmarks.

SAT.[5] As can be seen, SIMPSAT is the most robust among the four methods. It is intriguing that SIMPSAT outperforms CMS$^+$ by a substantial margin on several examples. Taking the extreme case `crc32-dat48` for example, SIMPSAT finished within 3 seconds while all other methods timed out at 7,200 seconds. A close investigation revealed that SIMPSAT was able to deduce from Gaussian elimination many more powerful short XOR-clauses (with lengths less than or equal to 2) than CMS$^+$ as seen from columns six and nine, where the numbers of XOR-clauses of lengths less than or equal to 2, denoted "#2xcl," are shown. These short XOR-clauses contributed to the effectiveness of SIMPSAT.

Another experiment on the benchmarks from randomly generated 3-regular graphs [12] is shown in Table 3. The number of instances of each benchmark suite is shown in the second column; the number of solved instances (within a 7,200-second limit) is shown in the third, fifth, and seventh columns; the entire runtime for solving solvable instances is shown in the fourth, sixth, and eighth columns. SIMPSAT and CMS$^+$ achieved similar results.

To study interpolant generation, a prototype, named MINISAT-GE, was built upon MINISAT-p 1.14 [10] (for which proof logging is supported) with XOR-constraint solving integrated as the pseudo code sketched in Figure 1. Benchmarks were created from a subset of the unsatisfiable instances of Table 3 by

[5] The cec command of ABC exploits circuit structure similarities and logic synthesis methods for efficient equivalence checking [18].

Table 2. Performance Comparison on Equivalence Checking of CRC Circuits.

Instance	ABC cec	CMS$^-$	CMS$^+$			SimpSat			Spdup over ABC	Spdup over CMS$^+$
	Time (sec)	Time (sec)	Time (sec)	GE Util (%)	GE #2xcl	Time (sec)	GE Util (%)	GE #2xcl		
`crc16-dat16`	0.11	0.04	0.02	23.37	1	0.02	33.03	15	6.88	1.38
`crc16-dat24`	0.53	0.17	0.16	26.79	1	0.04	49.53	16	12.93	4.00
`crc16-dat32`	1.44	1.77	2.05	10.58	2	0.11	33.27	15	12.97	18.48
`crc24-dat64`	4667.42	>7200	>7200	0.27	0	360.64	3.36	18	12.94	-
`crc24-dat64-only-flat`	31.52	>7200	>7200	0.17	0	4.71	36.30	22	6.69	-
`crc24-zer64-flat`	0.62	>7200	29.10	9.39	5	17.13	13.08	11	0.04	1.70
`crc24-zer64x2-flat`	0.51	498.17	633.49	3.47	0	0.73	16.42	19	0.69	863.20
`crc24-zer64x3-flat`	0.45	26.84	49.07	0.54	0	0.65	15.31	19	0.70	75.85
`crc32c-dat32`	596.94	>7200	>7200	22.34	0	0.22	47.95	35	2713.78	-
`crc32c-dat64`	>7200	>7200	>7200	0.00	7	3386.20	0.14	32	-	-
`crc32c-dat64-only`	1055.18	>7200	>7200	33.10	3	486.81	20.57	32	2.17	-
`crc32c-zer64`	0.86	101.39	102.70	4.93	5	0.54	28.78	34	1.59	190.21
`crc32-dat16`	0.91	7.21	6.88	10.56	1	0.49	56.18	31	1.85	14.02
`crc32-dat24`	2.51	64.94	10.70	30.86	3	0.93	63.11	32	2.71	11.56
`crc32-dat32`	385.73	>7200	2153.89	3.38	1	0.49	63.17	32	792.18	4423.46
`crc32-dat40`	6666.37	>7200	>7200	6.67	0	0.59	48.28	32	11339.10	-
`crc32-dat48`	>7200	>7200	>7200	17.01	4	2.23	57.09	32	-	-
`crc32-dat56`	>7200	>7200	>7200	23.36	0	146.22	1.12	32	-	-
`crc32-dat8`	0.21	0.40	0.40	0.00	0	0.40	0.00	0	0.52	1.00

Table 3. Performance Comparison on 3-Regular Graph Benchmarks.

Instance	#inst	CMS$^-$		CMS$^+$		SimpSat	
		#solved	Time (sec)	#solved	Time (sec)	#solved	Time (sec)
`mod2-rand3bip-sat`	165	103	136064.80	165	6.98	165	7.13
`mod2-rand3bip-unsat`	75	75	72.46	75	15.65	75	15.66
`mod2c-rand3bip-unsat`	75	75	962.40	75	871.05	75	862.89
`mod2-3cage-unsat`	23	23	18.00	23	15.93	23	15.95
`mod2c-3cage-unsat`	23	23	115.44	23	106.27	23	102.06

evenly assigning clauses to ϕ_A and ϕ_B for interpolation. Table 4 compares the interpolants generated from the refutation proofs of MiniSat using McMillan's clause interpolation rules and those generated from MiniSat-GE using our derivation rules. A 300-second limit was imposed on SAT solving, and interpolants were synthesized using ABC script `dc2`, `dc2`, `balance`. The so generated interpolants were compared in terms of their number of inputs, number of AIG (and-inverter graph) nodes, and number of logic levels. The reported runtime includes SAT solving time and interpolant synthesis time. In the table, an entry "-" indicates data unavailable due to timeout, or due to large interpolant sizes not practically synthesizable by ABC. As can be seen, XOR-constraint solving is effective in reducing SAT solving time and admits compact interpolant generation.

6 Related Work

Prior efforts [4, 15–17] deployed inference rules for XOR-reasoning. In [4], the authors proposed a framework integrating XOR-reasoning with the DPLL procedure

Table 4. Results on Interpolant Generation.

Instance	MiniSat					MiniSat-GE				
	#in	#node	#level	Time SAT (sec)	Time Syn (sec)	#in	#node	#level	Time SAT (sec)	Time Syn (sec)
`mod2-rand3bip-unsat-105-1`	45	32106	2273	4.32	23.58	45	132	14	0.01	0.1
`mod2-rand3bip-unsat-120-1`	-	-	-	88.93	-	44	129	12	0.01	0.12
`mod2-rand3bip-unsat-135-1`	-	-	-	280.31	-	54	159	14	0.01	0.09
`mod2-rand3bip-unsat-150-1`	-	-	-	53.45	-	50	147	14	0.01	0.09
`mod2-rand3bip-unsat-90-1`	34	111625	9730	0.63	388.28	34	99	12	0.01	0.09
`mod2c-rand3bip-unsat-105-15`	-	-	-	63.15	-	57	105	12	0.01	0.1
`mod2c-rand3bip-unsat-90-15`	30	105500	9375	1.56	175.32	31	5405	538	0.03	25.02
`mod2c-3cage-unsat-11`	-	-	-	>300	-	70	1857	137	0.01	1.55
`mod2-3cage-unsat-9-1`	-	-	-	74	-	26	75	12	0.01	0.09
`mod2-3cage-unsat-10-1`	-	-	-	>300	-	29	84	12	0.01	0.05

using Gauss resolution rules. However there was no implementation provided. In [15], the author focused on recognizing binary and ternary XOR-clauses for equivalence reasoning. Several inference rules were integrated into the DPLL search procedure for literal substitution. Based on the framework of [4], prior work [16, 17] proposed some lightweight inference rules for practical XOR-reasoning and supported with conflict-driven learning for XOR-clauses. The DPLL and XOR-reasoning procedures were integrated in a way similar to SMT solvers.

Compared to the closest prior work [26], our approach is similar but with the following main differences. For matrix representation, ours is in a reduced row echelon form, in contrast to the prior row echelon form. For matrix update, ours uses two-variable watching for incremental matrix update, in contrast to the prior column search and row swap. For matrix size, ours maintains a single-sized matrix for propagation/conflict detection, in contrast to the prior doubled-sized matrix. On the other hand, interpolant generation is supported in this work but not previously.

7 Conclusions and Future Work

Boolean satisfiability solving integrated with Gauss-Jordan elimination has been shown powerful in solving hard real-world instances involving XOR-constraints. With two-variable watching and simplex-style matrix update, Gauss-Jordan elimination has been made fast for complete detection of XOR-inferred implications/conflicts. Moreover, Craig interpolation has been made straight for compact interpolant generation, thus bypassing blind and unnecessarily detailed resolutions. For future work, extension to three-variable watching is planned for variable (in)equivalence, in addition to implication and conflict, detection.

Acknowledgments

The authors are grateful to Alan Mishchenko and Sayak Ray for providing the CRC benchmark circuits. This work was supported in part by the National

Science Council under grants NSC 99-2221-E-002-214-MY3, 99-2923-E-002-005-MY3, and 100-2923-E-002-008.

References

1. *Altera Advanced Synthesis Cookbook*: `http://www.altera.com/literature/manual/stx_cookbook.pdf`
2. A. Biere, M. Heule, H. van Maaren, and T. Walsh (editors). *Handbook of Satisfiability*, IOS Press, 2009.
3. Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification*: `http://www.eecs.berkeley.edu/~alanmi/abc/`
4. P. Baumgartner and F. Massacci. The taming of the (X)OR. In *Proc. Int'l Conf. on Computational Logic*, pages 508-522, 2000.
5. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient generation of Craig interpolants in satisfiability modulo theories. *ACM Trans. on Computational Logic*, 12(1): 7, 2010.
6. J.-C. Chen. XORSAT: An efficient algorithm for the DIMACS 32-bit parity problem. *CoRR abs/cs/0703006*, 2007.
7. J.-C. Chen. Building a hybrid SAT solver via conflict-driven, look-ahead and XOR reasoning techniques. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, pages 298-311, 2009.
8. W. Craig. Linear reasoning: A new form of the Herbrand-Gentzen theorem. *J. Symbolic Logic*, 22(3): 250-268, 1957.
9. G. Dantzig. Maximization of linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, pages 339-347, 1951.
10. N. Eén and N. Söensson. An extensible SAT-solver. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, pages 502-518, 2003.
11. C. Gomes, A. Sabharwal, and B. Selman. Model counting: A new strategy for obtaining good bounds. In *Proc. National Conf. on Artificial Intelligence (AAAI)*, pages 54-61, 2006.
12. H. Haanpaa, M. Jarvisalo, P. Kaski, and I. Niemela. Hard satisfiable clause sets for benchmarking equivalence reasoning techniques. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4): 27-46, 2006.
13. J.-H. R. Jiang, C.-C. Lee, A. Mishchenko, and C.-Y. Huang. To SAT or not to SAT: Scalable exploration of functional dependency. *IEEE Trans. on Computers*, 59(4):457-467, April 2010.
14. H.-Y. Liu, Y.-C. Chou, C.-H. Lin, and J.-H. R. Jiang. Towards Completely Automatic Decoder Synthesis. In *Proc. Int'l Conf. on Computer Aided Design (ICCAD)*, pages 389-395, 2011.
15. C. M. Li. Integrating equivalency reasoning into Davis-Putnam procedure. In *Prof. National Conf. on Artificial Intelligence (AAAI)*, pages 291-296, 2000.
16. T. Laitinen, T. Junttila, and I. Niemela. Extending clause learning DPLL with parity reasoning. In *Proc. European Conference on Artificial Intelligence (ECAI)*, pages 21-26, 2010.
17. T. Laitinen, T. Junttila, and I. Niemela. Equivalence class based parity reasoning with DPLL(XOR). In *Proc. Int'l Conf. on Tools with Artificial Intelligence (ICTAI)*, pages 649-658, 2011.
18. A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een. Improvements to combinational equivalence checking. In *Proc. Int'l Conf. on Computer-Aided Design (ICCAD)*, pages 836-843, 2006.

19. K. McMillan. Interpolation and SAT-based model checking. In *Proc. Int'l Conf. on Computer Aided Verification (CAV)*, pages 1-13, 2003.
20. K. McMillan. An interpolating theorem prover. *Theoretical Computer Science*, 345(1): 101-121, 2005.
21. F. Massacci and L. Marraro. Logical cryptanalysis as a SAT-problem: Encoding and analysis. *Journal of Automated Reasoning*, 24(1-2): 165-203, 2000.
22. M. Moskewicz, C. Madigan, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. Design Automation Conf. (DAC)*, pages 530-535, 2001.
23. J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Computeres*, 48(5): 506-521, 1999.
24. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6): 937-977, 2006.
25. M. Soos, K. Nohl, and C. Castelluccia. Extending SAT solvers to cryptographic problems. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, pages 244-257, 2009.
26. M. Soos. Enhanced Gaussian elimination in DPLL-based SAT solvers. In *Proc. Pragmatics of SAT*, 2010.
27. J. Warners and H. van Maaren. A two-phase algorithm for solving a class of hard satisfiability problems. *Operations Research Letters*, 23(3-5): 81-88, 1998.
28. G. Yorsh and M. Musuvathi. A combination method for generating interpolants. In *Proc. Int'l Conf. on Automated Deduction (CADE)*, pages 353-368, 2005.