The Taming of the (X)OR

Peter Baumgartner[1] and Fabio Massacci[2]*

[1] Institut für Informatik, Universität Koblenz-Landau
D-56073 Koblenz, Germany
peter@uni-koblenz.de http://www.uni-koblenz.de/~peter/
[2] Dip. di Ingegneria dell'Informazione
Università di Siena, 53100 Siena, Italy
massacci@dii.unisi.it, http://www.dii.unisi.it/~massacci/

Abstract. Many key verification problems such as bounded model-checking, circuit verification and logical cryptanalysis are formalized with combined clausal and affine logic (i.e. clauses with xor as the connective) and cannot be efficiently (if at all) solved by using CNF-only provers.
We present a decision procedure to *efficiently* decide such problems. The Gauss-DPLL procedure is a tight integration in a unifying framework of a Gauss-Elimination procedure (for affine logic) and a Davis-Putnam-Logeman-Loveland procedure (for usual clause logic).
The key idea, which distinguishes our approach from others, is the full interaction bewteen the two parts which makes it possible to maximize (deterministic) simplification rules by passing around newly created unit or binary clauses in either of these parts. We show the correcteness and the termination of Gauss-DPLL under very liberal assumptions.

1 Introduction

In many application areas such as formal verification [Cla90,BCC+99], logical cryptanalysis [Mas99,MM00], planning [KS96,GMS98], and AI in general [SKM97] the traditional formulation of a logical inference problem as a satisfiability problem in clausal normal form (CNF) is becoming unsatisfactory.

"Real world" problems are seldomly formulated in CNF and must always be converted to it. The natural formulations of real problems make use of many logical connectives: definitions (e.g. gates in circuit verifications), exclusive or (e.g. Feistel-operations in logical cryptanalysis), disjunctions (e.g. non-deterministic actions in planning) etc.

When such formulae are transformed into CNF the performance of the system is not very impressive, unless special heuristic information on the problem domain is used (see e.g. [GMS98] on planning and [WvM99] on the DIMACS parity bit problems).

Our motivating application was logical cryptanalysis, the encoding of cryptographic problems as SAT problems [Mas99,MM00]. Known plaintext attacks to the US Data Encryption Standard can be encoded as a SAT problem with formulae of increasing complexity. The experimental analysis in [Mas99,MM00] showed that the performance of state-of-the-art CNF solvers such as rel_sat [BS97], sato [Zha97], ntab [CA96], and

* F. Massacci acknowledges the support of a STM CNR grant.

J. Lloyd et al. (Eds.): CL 2000, LNAI 1861, pp. 508–522, 2000.

satz [Li99] degraded as soon as formulae containing exclusive-or appeared in the original formulation. Thus, solving real crypto-problems with CNF-provers looks unlikely.

A similar situation is found in circuit verification where the usage of successful BDD-packages [BRB90] has proven to be utterly ineffective when coping with fairly basics circuits such as multipliers. Parity bit problems, based on logically simple formulae, proved to be extremely hard for CNF based provers [SKM97,WvM99,JT96,Li00].

"The taming of the xor" has therefore become one of the major research efforts in the SAT community to tackle real world applications. The first solution is to cast the problem into CNF using advanced translations beyond Teitsin definitional translation [Wil90,GW00]. Otherwise one can use a dual-phase algorithm that solves the xor-part separately [WvM99], or more complex algorithms using multiple polynomials [WvM00]. Other researchers have focused on direct handling of xors as a black box subroutine of classical DPLL algorithms [Li00]. In the BDD community a number of "*DD" (where "*" may be instantiated to almost any alphabetic string) decision diagrams has been proposed to solve this problem [BDW95,DBR97].

Most of these works start from the observation that satisfiability of affine logic – sets of xor-clauses, i.e. clauses made up with xor as the connective – can be decided in polynomial time [Sch78]. In particular, one can use a Gaussian Elimination procedure (GE procedure) to decide a given affine logic problems in quadratic time.

It seems therefore possible to include GE as a black-box subroutine in a procedure for a more general logic, and this is indeed done in [Li00,WvM99,WvM00]. We will not directly do so, because the problems in our application domain (logical cryptanalysis), are beyond affine clause logic: after an appropriate transformation we end up with *two* sets of clauses, a set of usual or-clauses, and a set of xor-clauses. Our task is to decide the satisfiability of the combined problem, and, if satisfiable, output a model.

The experimental analysis reported by [Li00,WvM99,WvM00] showed that incorporating the GE procedure as a black-box subroutine definitely pays off if the affine logic part is overwhelming. This is the case for artificial DIMACS problems such as the bit parity problem or Pretolani's encoding of Urquhart's formulae [JT96,Li00,WvM99]. However, they also all agree that this is not sufficient when the affine logic part is only a part of the overall formula. This is indeed the case for DES encodings whereas xor clauses are just the hard core part (4% of whole) [Mas99,Li00]. This is true for many other problems such as model checking [BCC$^+$99,Li00].

So we want to have affine-logic reasoning in our calculus and, at the same time, we do not want to abandon the good, old and after all extremely efficient DPLL procedure. Our contribution is a revised DPLL where or-clauses and xor-clauses mutually co-exist.

In order to achieve a homogeneous architecture, we treat xor-clauses by more traditionally styled inference rules. In this way, the inferences carried out on either or-clauses or xor-clauses can heavily influence each other. This allows for performance optimizations by passing around newly created unit or binary clauses in the or-clause logic part to the xor-clause logic part (and vice versa). In both parts they can be used to *simplify* the currently derived clauses. By giving preference to simplifications, branching of the search space due to the or-logic part is delayed until unavoidable or even prevented.

Of course, our inference-rule based mechanism specializes to a variant of the GE procedure when restricted to affine logic. When applied to a pure inclusive-or clause

logic problem, the method instantiates to the (propositional version of the) well-known Davis-Putnam-Logeman-Loveland (DPLL) procedure [DLL62]. This choice is motivated by the nice properties of DPLL: its conceptual simplicity, space efficiency, few inference rules, efficient and adaptable implementations (the most efficient systematic propositional methods are based on DPLL [BS97,Zha97,CA96,Li99]), and the possibility to immediately extract a model in case that no refutation exists.

The suggested calculus in this paper can also be understood as an attempt to "lift" these properties to the case of a combined inclusive/exclusive-or logic. The underlying inference rules can roughly be divided in three classes: *Resolution*-type inferences to implement GE (however, one parent clause is *always* deleted), *simplification* inference rules (which do not cause branching) and the *cut* rule (aka split) to force a case analysis A – $\neg A$ to advance the derivation when other rules are no longer applicable.

This is only part of the story: one major difference between ours and the classical DPLL procedure is that we do not insist on explicitly computing a model; instead, we allow our procedure to terminate earlier, once a *functional description of a model* is computed. For instance, an equivalence like $A \equiv B$ is not subject to further case analysis to actually compute truth assignments for A and B. Instead it serves a functional description of our model. If we really want to have a truth assignment, we can choose a random value for, say, B and then the value of A can be easily calculated.

The rest of this paper is structured as follows: we start with some preliminary definitions. Then we introduce the basic ingredients of our calculus (simplification and GE inference rules). These are then combined with some more inference rules in a single calculus called Gauss-DPLL. Finally, we sketch its correctness.

2 Preliminaries

We apply the usual notions of propositional logic, in a way consistent to [CL73].

An *atom* is either a propositional variable or the symbol $\top$ ("*true*"). A *literal* is an atom or a negated atom. For a literal L, its *complement* $\overline{L}$ is the atom A, if one has $L = \neg A$, or else $\overline{L}$ is $\neg L$. For a literal L we denote by $|L|$ the atom of L, i.e. $|A| = A$ and $|\neg A| = A$ for any atom A.

An *assignment* is a pair A/L, where A is an atom different from $\top$, and L is a literal.

An *or-clause* is a possibly empty multiset $\{L_1, \ldots, L_n\}$ of literals, usually written as $L_1 \vee \cdots \vee L_n$ if $n > 0$, and $\Box$ if $n = 0$. Similarly, a *xor-clause* is a possibly empty multiset $\{L_1, \ldots, L_n\}$ of literals, usually written as $L_1 \oplus \cdots \oplus L_n$ if $n > 0$, and $\Box$ if $n = 0$. The atoms of a clause C, denoted by $|C|$ are computed in the obvious way as $|C| = \{|L| : L \in C\}$. A *clause* refers to an or-clause or a xor-clause.

Remark 1 (Special Cases). A clause with exactly one literal (i.e. a unit clause) can be seen as an or-clause, as a xor-clause or as an assignment where the value $\top$ or $\neg\top$ is assigned to the atom of the literal according the sign of the literal in the obvious way. A xor-clause with two literals can also be seen as an assignment. For instance $\neg A$ can be seen as the assignment $A/\neg\top$, whereas $A \oplus \neg B$ can be seen as the assignment B/A or the assignment A/B. The calculus below contains rules for such transitions.

In the sequel we use $A, B, \ldots$ for atoms, $K, L, \ldots$ for literals and $C, D, \ldots$ for clauses. The calligraphic letters $\mathcal{A}$, $\mathcal{C}$ and $\mathcal{X}$ are reserved to denote sets of assignment, sets of

or-clauses, and sets of xor-clauses, respectively. When writing down or-clauses, we use the notation $L \vee C$ to denote $\{L\} \vee C$, and $C \vee D$ to denote $C \cup D$ (and similarly for xor-clauses by using $\oplus$). Also, we write "$C,\ C$" instead of "$\{C\} \cup C$", where C is a (x)or-clause set.

Literal occurrences in xor-clauses can be flagged as *selected*. Selection is indicated by underlining, as in $\underline{L} \oplus C$. The purpose is to state C as a "definition" of L.

Quite frequently, we need the *set of selected atoms of* X, which is $\mathrm{sel}(X) = \{|L| \mid L \text{ is selected in } C, \text{ for some } C \in X\}$.

Translation to normal form. We have a strict separation in our clause sets: in the one part, only "$\vee$" occurs, and in the other part only "$\oplus$" occurs. Treating arbitrary propositional formulae is conceivable as well. However, due to the presence of xor-clauses, we can transform the initial formula into two separate sets, in a much simpler way than with CNF transformations.

For instance the formula $A \vee B \vee (C \oplus D \oplus E)$ can be transformed into the two clauses $A \vee B \vee F$ and $\neg F \oplus C \oplus D \oplus E$ introducing the new symbol F. It is easy to see that this is a satisfiability preserving transformation. Even with optimized CNF transformation we cannot get away with less than 6 clauses.

In our target application [MM00] we have only formulae of the form $L \leftrightarrow L_1 \oplus \cdots \oplus L_n$ or $L = L_1 \vee \cdots \vee L_n$. So, a transformation into normal form will be definitely easy. Hence we assume as given a clause-normal-form transformation that transforms the given formula φ (containing arbitrary connectives) equivalently into a set of or-clauses and a set of xor-clauses (read conjunctively).

3 Simplification by Boolean Reduction

Simplification by boolean reduction means to transform a clause into normal form by exploiting trivial boolean reductions. This is achieved by the inference rules R_{Bool} shown in Figure 1; they are also used in the preprocessing step of the encoding of DES in [MM00], and they extend to the xor-case the rules given in [Mas98,HS98].

More precisely, *reduction of a clause C by the R_{Bool} inference rules* means to repeatedly replace C by the result of a single application of an inference rule from R_{Bool} to C, resulting finally in a normal form of C.

Proposition 1. *The reduction of a clause C by the R_{Bool} inference rules terminates.*

The proof is straightforward and is omitted (the proof of Lemma 2 below makes the ordering explicit that guarantees termination).

Remark 2 (Transparent Selection). In the reduction process, the inference rules are applied to xor-clauses *transparently* wrt. selected literals according to the following rules: (i) selection within C (referring to the actual instance of the meta-variable in the inference rules R_{Bool}) is preserved. (ii) selection of L, A, $\neg A$, B or $\neg B$ carries over to the resulting clause, if the respective literal still is present (in complemented form, however) in the conclusion.

The reason to preserve selection is to make in the calculus a re-orientation of a definition impossible, where e.g. $\neg A$ is just as good a definition name as A.

Elimination of logical constants

$$L \oplus \top \oplus C \to \overline{L} \oplus C$$
$$\neg\top \oplus C \to C$$
$$\top \vee C \to \top$$
$$\neg\top \vee C \to C$$

Elimination of redundancies

$$L \oplus L \oplus C \to C$$
$$A \oplus \neg A \oplus C \to \top \oplus C$$
$$\neg A \oplus \neg B \oplus C \to A \oplus B \oplus C$$
$$L \vee L \vee C \to L \vee C$$
$$A \vee \neg A \vee C \to \top$$

Fig. 1. The inference rules R_{Bool} for boolean reduction; in "$\varphi \to \psi$" the left hand side φ is the premise and the right hand side ψ is the conclusion. The case $C = \Box$ is permitted in all rules, except of $\top \vee C \to \top$.

In general, a normal form derived in the way just described is not unique. Still, all normal forms are logically equivalent and this is what we are interested in. Thus we let $C\downarrow$ denote some arbitrary normal form of C.

For instance, $\underline{\neg A} \oplus \neg B \oplus \neg C$ has three normal forms and $\underline{\neg A} \oplus \neg B \oplus \neg C\downarrow$ may be $\underline{A} \oplus B \oplus \neg C$ (notice how selection carries over). The single normal form of $\underline{A} \oplus A \oplus B$ is B (however, such cyclic definitions are impossible to construct in the calculus).

4 Simplification by Boolean Assignments

The device introduced here is comparable to the uniform substitution rule by Teitsin. It has been already introduced in [Mas98,HS98].

Remark 3 (Assumptions about Sets of Assignments). From now on, when considering a set A of assignments, we insist that whenever $A/L \in A$ and $A/K \in A$ then $L = K$ (functionality), and whenever $A/L \in A$ then $|L|/K \notin A$, for every literal K (idempotency).

Notice that idempotency guarantees in particular $A/A \notin A$ and $A/\neg A \notin A$.

Definition 1 (Simplification by Assignments). *The* simplification of a clause C by a set of assignments $A = \{A_1/L_1, \ldots A_n/L_n\}$, *denoted by* $C//A$, *is obtained by simultaneous substitution of each occurrence of* A_i *(resp.* $\neg A_i$*) in* C *by* L_i *(resp.* $\overline{L_i}$*), for* $1 \le i \le n$.

Simplification is applied transparently to selected literals in a "destructive" way: if A (or $\neg A$) is selected in a xor-clause C and a simplification $C//\{A/L, \ldots\}$ is performed, the literal occurrence L (resp. $\overline{L}$) in the resulting clause does not get selected.

Definition 2. *An atom* A *is* defined in a set of assignments A *iff* $A/L \in A$, *for some literal* L. *It is* undefined *iff it is not defined.*

Definition 3 (Extending a Set of Assignments). *Let* A *be a set of assignments and* A/L *be an assignment such that both* A *and* $|L|$ *are undefined in* A. *Then, the* extension of A by A/L, *denoted by* $A \circ (A/L)$, *is the set of assignments* $\{B/(K//\{A/L\}) \mid B/K \in A\} \cup \{A/L\}$. *In this definition the literal* K *is read as a unit clause.*

If the atom A *is undefined in* A, *then* $A \circ A = A \circ (A/\top)$, *and* $A \circ \neg A = A \circ (A/\neg\top)$

Lemma 1 (Preservation of Properties). *If A is functional and idempotent, then, under the conditions stated in Def. 3, both $A \circ (A/L)$ and $A \circ L$ are functional and idempotent.*

Proof. (Sketch) Consider the general case $A \circ (A/L)$. Since A is undefined in A and only the right hand sides are modified by extension, functionality is preserved. $A \circ (A/L)$ is idempotent because the right hand sides in A are subject to substitution by the new assignment A/L and that $|L|$ is undefined in A. This makes non-idempotency impossible.

5 Gauss Resolution Rules

The Gauss Elimination (GE) procedure can be represented by two resolution-like rules:

$$\mathsf{Gauss}^{-}\ \frac{L \oplus C \qquad \overline{L} \oplus D}{C \oplus D} \qquad \mathsf{Gauss}^{+}\ \frac{L \oplus C \qquad L \oplus D}{\top \oplus C \oplus D}$$

For the Gauss^{-} rule, we say that $C \oplus D$ *is the Gauss-resolvant of* $L \oplus C$ *on* L *into* $\overline{L} \oplus D$, and similarly for Gauss^{+} rule.

As for resolution, these rules are sound, i.e. the conclusion is a consequence of the premises. However, in sharp contrast to resolution, in both rules each premise is a consequence of the conclusion and the other premise:

Proposition 2. *All of the following hold:*

1. $\{L \oplus C,\ \overline{L} \oplus D\} \models C \oplus D$ *and* $\{L \oplus C,\ C \oplus D\} \models \overline{L} \oplus D$
2. $\{L \oplus C,\ L \oplus D\} \models \top \oplus C \oplus D$ *and* $\{L \oplus C,\ \top \oplus C \oplus D\} \models L \oplus D$

Using proposition 2 we can delete one of the premises once the Gauss-resolvant has been added to the xor-clause set *without loosing completeness*, because it is an equivalence preserving transformation. The intuition is that the deleted clause can always be restored by applying the inference rules. Thus one can avoid the exponential explosion of resolution: the number of clauses never grows more than the initial set of clauses. If we apply boolean reduction rules, one can eliminate duplicated literals in a clause, and hence the length of each clause never exceeds the number of available atoms.

These two rules, together with a deletion strategy, describe a Gauss-Elimination procedure as known from high-school which has a quadratic complexity. Take the given xor-clauses $X = \{C_1, \ldots, C_n\}$ as a system of linear equations in a boolean ring $\oplus C_1 = 1, \ldots, \oplus C_n = 1$, where each variable is assigned a value 0 or 1, $\oplus$ is addition modulo 2, and $\neg A$ is $A \oplus 1$. In this view, the overall strategy to determine whether X is satisfiable is first to derive (if possible) a triangular form of X. For this, select a clause with a literal, say L, and eliminate with the two rules all occurrences of L and $\overline{L}$ from the remaining clauses. This is possible by design of the inference rules, as the conclusion contains neither L nor $\overline{L}$. If necessary, we apply boolean reduction rules until each clause contains at most one occurrence of L or $\overline{L}$. Next, the clause containing L is put aside and the variable elimination process continues in this way until all clauses are processed.

If the empty clause comes up, the xor-clause set is unsatisfiable. If a triangular matrix results, a unique model can be computed by propagating the assignments forced

by the shorter clauses towards the longer clauses. For a non-triangular form, the system is under-determined and more than one model exists.

Unfortunately, unrestricted application of the Gauss$^-$ and Gauss$^+$ rules to a set of xor-clauses may be a non-terminating process. The system might cycle among a finite set of logically equivalent forms without reaching a fix point.

As an example consider $X = \{A \oplus C,\ \neg A \oplus D\}$. Resolving $A \oplus C$ on A into $\neg A \oplus D$ yields $X' = \{A \oplus C,\ C \oplus D\}$. Next, resolving $A \oplus C$ on C into $C \oplus D$ results in X again (after reduction).

This problem is solved by using the strategy described above to derive a triangular form. It would be acceptable if the xor-clause set is fixed, but this is not our case: first, new unary or binary xor-clauses may come up as the derivation proceeds, and it can be advantageous to delay the decision on the variables to eliminate. Second, the initial xor-clause set is undetermined in most cases [Li00], and the value of many "independent" variables is determined only by the constraints expressed by the or-clauses.

6 Gauss-DPLL

In this section we introduce the inference rules which are at the basis of a generalization of the Davis-Putnam-Logeman-Loveland Procedure, which we call *Gauss*-DPLL.

The inference rules, but one, are of the form

$$\text{Name}\ \frac{A \quad C \quad X}{A' \quad C' \quad X'}\ \text{Condition}$$

where A is a set of assignments, C is a set of or-clauses, X is a set of xor-clauses, possibly with some selected literals. The primed versions are the sets derived by the rule.

The intuition is that in A we store the definitions A/L which say how to set the value of an atom A on the basis of the value of another atom or a logical constant. The sets C and X contain the (x)or-clauses that have not been completely processed yet.

The main idea behind selected literals is that a xor-clause C containing a selected literal L can be seen as a definition of the corresponding atom $|L|$ in terms of the value of the other literals of C. For the whole system to be consistent, the clause C can only be used as the definition of *only one* atom. Further, the calculus achieves that there is only one such definition – be it in just one single xor-clause or as an assignment.

The twist to implement the GE procedure in this way is, that, when no rule is applicable, the set X and the selected literals in it implicitly describe a triangular form of the linear modulo 2 equations in X. For instance, if $X = \{\underline{A} \oplus B,\ \underline{C} \oplus B\}$ this implicitly describe a triangular form which is (partly) undetermined: A and C have been "solved" as functions of B. In terms of linear equation this is obvious: we have two equations and three variables.

We are now turning to the inference rules of *Gauss*-DPLL.

The following inference rules are used to reduce clauses; to avoid trivial loops, the applicability condition $C \neq C\downarrow$ is assumed:

$$\vee\text{-Red}\ \frac{A \quad C, C \quad X}{A \quad C\downarrow, C \quad X} \qquad \oplus\text{-Red}\ \frac{A \quad C \quad C, X}{A \quad C \quad C\downarrow, X}$$

The following inference rules simplify a clause by the current assignments; the applicability condition $C \neq C//A$ is assumed:

$$\vee\text{-Simp}\ \frac{A \quad C,\ C \quad X}{A \quad C//A,\ C \quad X} \qquad \oplus\text{-Simp}\ \frac{A \quad C \quad C,\ X}{A \quad C \quad C//A,\ X}$$

An inference rule for the simplification of A wrt. A is not necessary, because A is both functional and idempotent (cf. Remark 3) as being constructed.

Now we turn to the inference rules to implement the GE procedure as described in Section 5. First, we need a rule to select a literal L for elimination.

$$\text{Select}\ \frac{A \quad C \quad L \oplus C,\ X}{A \quad C \quad \underline{L} \oplus C,\ X} \quad \begin{cases} \text{if } \mathrm{sel}(\{L \oplus C\} \cup X) \cap \mathrm{atoms}(L \oplus C) = \{\} \\ \text{and } |L| \neq \top \end{cases}$$

The intuition behind the applicability condition is that we can use a xor-clause as definition of only one literal at a time, i.e. $\mathrm{sel}(L \oplus C) \cap \mathrm{atoms}(L \oplus C) = \{\}$. To guarantee that no trivially cyclic definition as in $\underline{A} \oplus A \oplus B$ comes up, the $\oplus$-Red inference rule must be preferred to Select (all the required preferences are stated in Def. 4 below). The subcondition $\mathrm{sel}(X) \cap \mathrm{atoms}(L \oplus C) = \{\}$ states that the new definition must not depend from other definitions. If it were absent, a cyclic situation as in $\{\underline{A} \oplus B,\ A \oplus \underline{\neg B}\}$ comes up easily.

Then we have the proper Gauss-Resolution rule:

$$\text{Gauss}\ \frac{A \quad C \quad \underline{L} \oplus C,\ D,\ X}{A \quad C \quad \underline{L} \oplus C,\ D',\ X} \quad \begin{cases} \text{if } D' \text{ is Gauss resolvent of } L \oplus C \\ \text{on } L \text{ into } D \end{cases}$$

Intuitively, this rule says that we take $\underline{L} \oplus C$ as a definition of L and replace in D the literal L (or $\overline{L}$) by its definition. To guarantee that there is no occurrence of L (or $\overline{L}$) left in D', the $\oplus$-Red inference rule must be preferred to Gauss. The Gauss rule is applied transparently wrt. selected literals, i.e. a possibly selected literal in D remains selected in D' (the literal L (or $\overline{L}$) in D' cannot be selected anyway, cf. invariant (ii) in Lemma 2).

Example 1. Consider the following derivation where A and C have been removed for readability and numbers are for reference:

(1)	$A \oplus B \oplus E,\ A \oplus C,\ B \oplus C$	start
(2)	$A \oplus B \oplus E,\ \underline{A} \oplus C,\ B \oplus C$	by Select
(3)	$C \oplus B \oplus E \oplus \top,\ \underline{A} \oplus C,\ B \oplus C$	by Gauss$^+$ of $A \oplus C$ on A into $A \oplus B \oplus E$
(4)	$\neg C \oplus B \oplus E,\ \underline{A} \oplus C,\ B \oplus C$	by $\oplus$-Red on $C \oplus B \oplus E \oplus \top$
(5)	$\neg C \oplus B \oplus E,\ \underline{A} \oplus C,\ B \oplus \underline{C}$	by Select
(6)	$B \oplus B \oplus E,\ \underline{A} \oplus C,\ B \oplus \underline{C}$	by Gauss$^-$ of $B \oplus C$ on C into $\neg C \oplus B \oplus E$
(7)	$E,\ \underline{A} \oplus C,\ B \oplus \underline{C}$	by $\oplus$-Red on $B \oplus B \oplus E$
(8)	$E,\ \underline{A} \oplus B \oplus \top,\ B \oplus \underline{C}$	by Gauss$^+$ of $B \oplus C$ on C into $A \oplus C$
(9)	$E,\ \underline{A} \oplus \neg B,\ B \oplus \underline{C}$	by $\oplus$-Red on $\underline{A} \oplus B \oplus \top$
(10)	$\underline{E},\ \underline{A} \oplus \neg B,\ B \oplus \underline{C}$	by Select

Now we can apply neither Gauss, nor Select, and indeed we terminated with an undetermined set of equations where A and C are defined in terms of B.

This example explains well the importance of giving precedence to the $\oplus$-Red rule over the Gauss rule. Consider step (6): without simplifying the two Bs we will not be able to eliminate them: Gauss alone will introduce two Cs, or two As etc.

To see the importance of the applicability condition of Select, let us look at the last step. Without it, we could have continued as follows:

$$\begin{array}{ll} (11') & E,\ \underline{A}\oplus\neg B,\ \underline{B}\oplus\underline{C} \quad \text{bySelect} \\ (12) & E,\ \underline{A}\oplus\underline{C},\ \underline{B}\oplus\underline{C} \quad \text{byGauss}^- \\ (13) & E,\ \underline{A}\oplus\underline{B}\oplus\top,\ \underline{B}\oplus\underline{C} \quad \text{byGauss}^+ \\ (14) & E,\ \underline{A}\oplus\underline{\neg B},\ \underline{B}\oplus\underline{C} \quad \text{by}\oplus\text{-Red} \end{array}$$

So we are using $A\oplus C$ sometimes as a definition of A and sometimes as a definition of C. This will clearly lead to a non-terminating sequence.

The rules presented so far constitute the core of the GE procedure as described in Section 5. The next set of inference rule transforms unit (x)or-clauses into assignments, with the purpose to trigger new simplification steps.

$$\vee\text{-Unit}\ \frac{A \qquad L,\ C \qquad X}{A\circ L \qquad C \qquad X} \qquad \oplus\text{-Unit}\ \frac{A \qquad C \qquad L,\ X}{A\circ L \qquad C \qquad X}$$

Here, L may or may not be selected.

Remark 4. Since we give preference to Red and Simp over the Unit rules, the extension of A to $A\circ L$ is defined, i.e. $|L|$ is undefined in A. Thus, functionality and idempotency are preserved by Lemma 1, and the set of assignments strictly increases.

Now, the well-known DPLL splitting rule is introduced. The purpose is to advance a derivation once no other rule is applicable.

$$\text{Split}\ \frac{A \quad C \quad X}{A\circ A \quad C \quad X \qquad\qquad A\circ\neg A \quad C \quad X} \quad \text{if } A\in\text{atoms}(C), \text{ for some } C\in\mathcal{C}$$

This is the sole rule with two consequences. Observe that the splitting in the two cases A and $\neg A$ is expressed in our notation as two respective assignments $A/\top$ and $A/\neg\top$.

Remark 5. Once again we have no condition such as "A is undefined in A" because we give preference to Red and Simp over Split, and therefore the same reasoning as in Remark 4 applies.

The applicability condition in Split is not necessary for completeness, but is useful for stopping the search without going to compute explicitly any of the models that would be possible by assigning all combinations of $\top$ and $\neg\top$ to the "independent" atoms occurring in definitions represented by the X. However, for the atoms occurring in C, applying Split is mandatory as the last resort to make progress in processing C.

Remark 6 (Explicit Models). If we arrive at a stage where no rule is applicable, and the empty clause has not been found, we have a functional description of a model. To obtain a model as a set of assignments of logical constants to atoms, we can add to A an arbitrary truth value assignment for each atom that is not selected in a xor-clause in X. The exhaustive application of the Simp, Red and Unit rules leads to the desired result in A then.

The next rules for equivalences are not necessary for completeness but they allow for a substantial speed-up as they correspond to powerful forms of pruning: some hard DIMACS problems are solved by using rules of equivalent form alone in [Li00].

$$\vee\text{-Eqv-1}\ \frac{A \qquad A \vee B,\ \neg A \vee \neg B,\ C \qquad X}{A \circ (A/\neg B) \qquad C \qquad X}\ \text{if } B \notin \mathrm{sel}(X)$$

$$\vee\text{-Eqv-2}\ \frac{A \qquad A \vee \neg B,\ \neg A \vee B,\ C \qquad X}{A \circ (A/B) \qquad C \qquad X}\ \text{if } B \notin \mathrm{sel}(X)$$

$$\oplus\text{-Eqv}\ \frac{A \qquad C \qquad A \oplus L,\ X}{A \circ (A/\overline{L}) \qquad C \qquad X}\ \text{if } |L| \notin \mathrm{sel}(\{A \oplus L\} \cup X)$$

Remark 7. Similarly as said above in Remark 4 for the Split rule, we insist to prefer the Simp and Red rules over the Eqv rules. Therefore, all stated extensions of A in the Eqv rules are defined, thus both functionality and idempotency are preserved (cf. Lemma 1), and also A is undefined in A.

To avoid loops, the turning of (x)or-clauses into assignments by the Eqv rules must not contradict the implicit ordering of literals as determined by the selected literals in X (this ordering is made explicit in the proof of Lemma 2). This is what the stated applicability conditions are good for.

The $\oplus$-Eqv rule is formulated general enough, because any binary xor-clause of the form $\neg A \oplus \neg B$ can be turned into $A \oplus B$ by boolean reduction.

7 An Effective Calculus for Proof Search

Finally, it has to be said how to combine the inference rules of Section 6:

Definition 4 (Affine Logic Tree (ALT)). *We consider (incomplete) binary trees where every node N is labelled with a tuple (A, C, X). The label of N is denoted by $\lambda(N)$.*

Affine logic trees, ALTs, for C and X, *where C (resp. X) is an or-clause set (resp. xor-clause set) are defined inductively in the following way:*

Initialization Step: *the tree T consisting of a root node N only and such that $\lambda(N) = (\{\}, C, X)$ is an ALT for C and X.*

Non-branching Extension Step: *if N' is a leaf of an ALT T' for C and X, and one of the non-branching inference rules is applicable to $\lambda(N')$, then T is an ALT for C and X, where T is obtained from T' by attaching one new child node N below N', and $\lambda(N)$ is obtained by a single application of one of the non-branching inference rules to $\lambda(N')$. Applicability of these inference rules is given preference as follows:*
- $\oplus$-Simp *and* $\oplus$-Red *must be applied before* Gauss *and* Select
- $\oplus$-Simp *and* $\oplus$-Red *must be applied before* $\oplus$-Unit *and* $\oplus$-Eqv
- $\vee$-Simp *and* $\vee$-Red *must be applied before* $\vee$-Unit, $\vee$-Eqv-1 *and* $\vee$-Eqv-2.

Branching Extension Step: *if N' is a leaf of an ALT T' for C and X, and non-branching extension steps are not applicable to N', and* Split *is applicable to $\lambda(N')$, then T is an ALT for C and X, where T is obtained from T' by attaching two new child nodes N_l and N_r below N', and $\lambda(N_l)$ and $\lambda(N_r)$ are obtained by a single application of* Split *to $\lambda(N')$.*

We abbreviate "ALT for C and X" as "ALT" if context allows.

Definition 5 (Open, Closed, Derivation, Finishedness, Fairness). *A branch B in an ALT T is* closed *iff for some node N of B it holds $\square \in C \cup X$, where $\lambda(N) = (A, C, X)$. Otherwise it is* open. *An ALT T is closed iff every branch of T is closed, otherwise it is* open. *The branch B is* finished *iff B is closed or no extension step is applicable to the leaf of B. An ALT T is finished iff every branch of T is finished. The term* unfinished *means not "not finished". A* derivation D (for given C and X) *is a sequence $T_0, T_1, \ldots, T_n, \ldots$ of ALTs, such that T_0 is obtained by an initialization step, and for $i > 0$, T_i is obtained by an extension step applied to T_{i-1}. A derivation D is* fair *iff it does not end in an unfinished ALT.*

Remark 8. The ALTs are the objects that are actually computed with. Observe that a *fair* derivation either ends in a closed ALT (which means that the set $C \cup X$ is unsatisfiable), or ends in an open ALT with at least one open and finished branch (which, as will be shown, represents a *functional description* of a model for $C \cup X$), or does not terminate (which will be shown to be impossible in Lemma 2).

An *effective proof procedure* can be constructed by the simplest greedy strategy: start with an ALT for C and X by an initialization step, and apply extension steps as long as possible. Thereby, one would actually pursue only one branch at a time, not further extend closed branches and delete closed branches from memory as soon as derived. Under these regime, only polynomial space is consumed.

We do not specify a sophisticated proof procedure here, in particular since the design of an *efficient* proof procedure that takes advantage of good strategies for the unspecified parameters (selection of literals, actual preference of inference rules) depends from practical experiments which have not been carried out yet. For instance, it seems natural to choose, among the possible selections of literals in xor-clauses, those that maximize the future application of the Eqv or Unit rules. Fortunately, the correctness proof in the next section guarantees that any setting within the inference rule preferences stated in Definition 4 is complete.

8 Correctness

The *soundness* proof – that any closed ALT for C and X indicates unsatisfiability of $C \cup X$ – is done by standard means and is omitted. To show *completeness*, we first show that exhaustive application of the inference rules always terminates:

Lemma 2 (Termination). *Any derivation D for given C and X is finite.*

Proof. It suffices to show that no branch can be endlessly extended. At the heart of this proof are well-founded, strict partial orderings $\succ_N$ on clauses associated to the nodes N of the constructed ALTs. As a preliminary step, let $>_N$ be a binary relation over atoms associated to node N, which is defined inductively as follows:

$$>_N = \begin{cases} \{(A, \top) \mid A \text{ is an atom}\}, \text{ if } N \text{ is the root node} \\ >_{N'} \cup \{(A/L) \mid (A/L) \in A\} \\ \qquad \cup \{(|K|, |L_1|), \ldots, (|K|, |L_k|) \mid \underline{K} \oplus L_1 \oplus \cdots \oplus L_k \in X\} \ , \\ \quad \text{where } \lambda(N) = (A, C, X), \text{ if } N \text{ has the immediate ancestor node } N'. \end{cases}$$

That is, $>_N$ starts in a trivial way, and gets enlarged as new assignments come up or selections are done when going down the branches. An important detail is that $>_N$ monotonically increases in this process.

The transitive closure of $>_N$ is denoted by $\succ_N$. In order to compare clauses take the usual multiset extension $\gg_N$ of the literal ordering in which L_1 is strictly greater than L_2 iff $|L_1| \succ_N |L_2|$ or else $L_1 = \neg L_2$ (i.e. $\neg A$ is greater than A). It is well-known that if $\succ_N$ is a strict, well-founded ordering (on atoms), as will be shown below, so is $\gg_N$ (on clauses).

We need several *invariants* to hold for each node N, where $\lambda(N) = (A, C, X)$:

(i) If $|K| >_N |L|$, then (a) $|K|$ is the left hand side of an assignment in A, or (b) K or $\overline{K}$ is the selected literal in some xor-clause in X.

(ii) For each selected atom $A \in \mathrm{sel}(X)$ there is exactly one xor-clause $C \in X$ such that A or $\neg A$ is a selected literal in C. Furthermore, this literal is the only selected literal occurrence in C.

(iii) If N has an immediate ancestor node N' and $\mathrm{sel}(X) \subset \mathrm{sel}(X')$, where $\lambda(N') = (A', C', X')$, then either the $\oplus$-Simp rule or the $\oplus$-Unit rule or the $\oplus$-Eqv rule is applied to N' to obtain N (but no other rule). That is, if a selection is lost, these are the only possible sources.

(iv) The relation $\succ_N$ is a strict partial ordering on atoms.

(v) If N has an immediate ancestor node N' and Select is applied to N', then $>_N \supset >_{N'}$.

The proof of the invariants is omitted here for space reasons; it can be found in the full version. They are used now to argue for termination. We feel no need for a completely formal presentation of the lexicographic ordering underlying the following argumentation.

Suppose, to the contrary, there is an infinite sequence of branches $B_0, B_1, \ldots$ such that, for $j \geq 0$, B_j is a branch of some T_i of the given derivation (written as in Def. 5), and that an extension step is applied to the leaf of B_j, and B_{j+1} is a branch resulting from this application. We are now investigating possible sources for this branch sequence to be infinite.

First, from some point in time on, the $>_N$-relation is the same (referring to the leaves of the branches in the considered branch sequence), because only finitely many literals are at disposal, and $>_{N_j} \subseteq >_{N_{j+1}}$ by construction of $>_N$, where N_j is the leaf of B_j.

Consequently, together with invariant (v), the Select rule is applied a *last* time along the considered branch sequence.

Second, each of $\vee$-Unit, $\oplus$-Unit, and Split is applied a last time, because each of them *strictly* increases the set of assignment it modifies. This was argued for in Remarks 4 and 5. Clearly, this strictness suffices as a proof for the claim.

Third, each Eqv rule is applied a last time. The arguments are the same is in "second", by using Remark 7.

Fourth, the rules mentioned at "second" and "third" are the only ones to extend assignments. Hence, from some point in time on, the set of assignments is the same in each leaf of the considered branch sequence.

Fifth, from "fourth" and the idempotency of assignments (cf. again Remarks 4, 5 and 7) it follows immediately that the Simp rules are applied a last time.

Sixth, hence, only the Gauss and Red rules remain as sources for infiniteness of the branch sequence. To show this impossibility, observe that with "first" the ordering $\gg_N$

is the same from some point in time on (invariant (iv) guarantees that $\succ_N$ is indeed a well-founded, strict partial ordering). Further, the ordering $\succ_N$ is made such that the Gauss and Red rules both work strictly decreasing. More precisely, the Gauss rule refers to the Gauss$^-$ and Gauss$^+$ rules. These are applied with a left premise, in which L is the only selected literal (cf. the applicability condition of Gauss and invariant (ii)) and which is strictly larger than each of the rest literals (by construction of the ordering). Hence, the right premise strictly decreases wrt. $\succ_N$. For the Red rules it is straightforward to check that they work strictly decreasing wrt. $\succ_N$, provided they are applicable. An important detail is to make $\neg A$ bigger than A.

Hence, in sum, with Gauss and Red working strictly decreasing wrt. $\succ_N$, which is the same from some point in time on, both of them are applied a last time.

All inference rules are now shown to be applied a last time along the considered branch sequence. Hence it must be finite, and thus the lemma is proven.

Theorem 1 (Completeness). *Let D be a fair derivation for a set of or-clauses C and a set of xor-clauses X. Then, D is finite, and if the last ALT T in D is open, then $C \cup X$ is satisfiable.*

This is our main result. Observe that in the contrapositive direction it just expresses refutation completeness.

Proof. Finiteness of D is given by Lemma 2. Therefore suppose that T is the last ALT in D, and that T is open. We are concentrating on an open and finished branch B in T, which must exist according to Remark 8. Let N be the leaf of B, and $\lambda(N) = (A, C, X)$. The first observation is that $C = \{\}$ or $C = \{\top\}$ (which is equivalent). The proof is by contradiction: the case that C contains the empty clause is impossible, because then B would be closed. Also, if C would consist of clauses containing the symbol $\top$ only (with the single exception of the clause $\top$), it would would have been simplified to either $C = \{\}$, $C = \{\top\}$ or $C = \{\Box\}$ (contradicting the finishedness of B). Hence C contains at least a clause with a literal different from $\top$. Let L be such a literal. But then, Split with $|L|$ would have been applied, contradicting finishedness of B again. This completes the proof that $C = \{\}$ or $C = \{\top\}$.

Thus, to construct a model, we have to consider A and X only. We use the strategy indicated in Remark 6: we give an arbitrary value to the variables that are not selected in X, and we show how to extend to a model.

Fact: in each clause $C \in X$ there is exactly one occurrence of a selected literal, and all the selected literals are pairwise different (modulo sign). This is due to invariant (ii) in the proof of Lemma 2, and the finishedness of B. For, if in some $C \in X$ no literal would be selected, and Select is not applicable to C, then some literal in C is selected in a different clause (modulo sign), and thus Gauss would be applicable, contradicting finishedness.

Now take any literal L occuring in X but such that $|L| \notin \mathrm{sel}(X)$. Add it as an assignment $|L|/\top$ (or $|L|/\neg\top$) to A. A must still be idempotent and functional, because as a consequence of finishedness, $|L|$ must be undefined in A, and so Lemma 1 is applicable. Repeat this, until all non-selected literals receive an explicit (arbitrary) truth value in A.

Finally, only the selected literals in X do not have explicit truth values in A. Since each of them occurs only once in a clause in X (by the above *fact*), their truth values can

be chosen locally to the containing clauses as the appropriate parity for the rest clause, which has been completely specified by the arbitrary assignments. Furthermore, by the *fact* again, this can be done for *every* xor-clause in X. Hence, for each such selected literal L and its appropriate truth value, add a respective assignment $|L|/\top$ (or $|L|/\neg\top$) to A. This is possible, because, by finishedness again, L must be undefined in A (by the $\oplus$-Simp rule). Finally, explicit truth assignments for all the literals occuring as right hind sides in A are added arbitrarily. This procedure results in a functional assignment to either $\top$ or $\neg\top$ for all atoms, which is just a model.

9 Conclusions

In this paper we have presented a decision procedure called Gauss-DPLL for combined clausal and affine logic (i.e. clauses with xor as the connective).

We have argued that procedures to solve such problems are needed to *efficiently* decide respective problems, which occur frequently in real-world applications like circuit verification and logical cryptanalysis. Gauss-DPLL is a tight integration in a unifying framework of a Gauss-Elimination procedure (for affine logic) and a Davis-Putnam-Logeman-Loveland procedure (for usual clause logic).

The main ideas, which distinguishes our approach from other approaches in the literature, are the following: at first, we provide a coherent approach of the treatment of both or and xor-clauses which specialized to optimized decision procedures when the input is restricted to either of them. Second we allow for a heavy interleaving of the two parts with the purpose to maximize (deterministic) simplification by passing around newly created unit or binary clauses in either of these parts. Last, but not least, we are able to stop the search and output a functional description of the model rather than a completely specified model.

As noted in [Li00], the explicit handling of equivalences makes it possible to transform exponentially long proofs of hard DIMACS benchmarks by Dubois and Pretolani [JT96,Li00] using classical DPLL into short polynomial proofs. This result is accomplished by Li using rules corresponding to restricted versions of boolean reduction, simplifications and equivalences. The Gauss-DPLL procedure also inherits that speed-up over classical DPLL.

The calculus is not implemented yet, but we plan to do so in the near future.

References

BCC+99. A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. of ACM/IEEE DAC-99*, pages 317–320. ACM Press, 1999.

BDW95. B. Becker, R. Drechsler, and R. Werchner. On the relation between BDDs and FDDs. *Inf. and Comp.*, 123(2):185–197, 1995.

BRB90. K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *Proc. of ACM/IEEE DAC-90*, pages 40–45. IEEE Press, 1990.

BS97. R. Bayardo and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proc. of AAAI-97*, pages 203–208. AAAI Press/The MIT Press, 1997.

CA96. J. Crawford and L. Auton. Experimental results on the crossover point in random 3SAT. *AIJ*, 81(1-2):31–57, 1996.

CL73. C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

Cla90. L. Claesen, ed. *Formal VLSI Correctness Verification: VLSI Design Methods*, volume II. Elsevier, 1990.

DBR97. R. Drechsler, B. Becker, and S. Ruppertz. Manipulation algorithms for K*BMDs. In *Proc. of TACAS-97*, LNCS 1217, pages 4–18. Springer-Verlag, 1997.

DLL62. M. Davis, G. Logeman, and D. Loveland. A machine program for theorem proving. *CACM*, 5(7), 1962.

GMS98. E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act and the rest will follow: Exploiting nondeterminism in planning as satisfiability. In *Proc. of AAAI-98*, pages 948–952. The MIT Press, 1998.

GW00. J. Groote and J. Warners. The propositional formula checker HeerHugo. *JAR*, 2000. To appear.

HS98. U. Hustadt and R. Schmidt. Simplification and backjumping in modal tableau. In *Proc. of TABLEAUX-98*, LNAI 1397 , pages 187–201. Springer-Verlag, 1998.

JT96. D. Johnson and M. Trick, eds. *Cliques, Coloring, satisfiability: the second DIMACS implementation challenge*, volume 26 of *AMS Series in Discr. Math. and TCS*. AMS, 1996.

KS96. H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stocastic search. In *Proc. of AAAI-96*, pages 1194–1201. The MIT Press, 1996.

Li99. Chu-Min Li. A constraint-based approach to narrow search trees for satisfiability. *IPL*, 71(2):75–80, 1999.

Li00. Chun-Min Li. Integrating equivalency reasoning into Davis-Putnam procedure. To appear in *Proc. of AAAI-00*.

Mas98. Fabio Massacci. Simplification: A general constraint propagation technique for propositional and modal tableaux. In *Proc. of TABLEAUX-98*, LNAI 1397, pages 217–231. Springer-Verlag, 1998.

Mas99. Fabio Massacci. Using Walk-SAT and Rel-sat for cryptographic key search. In *Proc. of IJCAI-99*, pages 290–295. Morgan Kaufmann, 1999.

MM00. Fabio Massacci and Laura Marraro. Logical cryptanalysis as a SAT-problem: Encoding and analysis of the u.s. Data Encryption Standard. *JAR*, 2000. To appear.

Sch78. T. Schaefer. The complexity of satisfiability problems. In *Proc. of STOC-78*, pages 216–226. ACM Press, 1978.

SKM97. Bart Selman, Henry Kautz, and David McAllester. Ten challenges in propositional resoning and search. In *Proc. of IJCAI-97*, pages 50–54. Morgan Kaufmann, 1997.

Wil90. J. Wilson. Compact normal forms in propositional logics and integer programming formulations. *Comp. and Op. Res.*, 17(3):309–314, 1990.

WvM99. J. Warners and H. van Maaren. A two phase algorithm for solving a class of hard satisfiability problems. *Op. Res. Lett.*, 23(3-5):81–88, 1999.

WvM00. J. Warners and H. van Maaren. Recognition of tractable satisfiability problems through balanced polynomial representations. *Discr. Appl. Math.*, 2000.

Zha97. H. Zhang. SATO: An Efficient Propositional Theorem Prover. In *Proc. of CADE 97*, LNAI 1249, pages 272–275, 1997. Springer-Verlag.