Conflict-Driven XOR-Clause Learning*

Tero Laitinen, Tommi Junttila, and Ilkka Niemelä

Aalto University
Department of Information and Computer Science
PO Box 15400, FI-00076 Aalto, Finland
{Tero.Laitinen,Tommi.Junttila,Ilkka.Niemela}@aalto.fi

Abstract. Modern conflict-driven clause learning (CDCL) SAT solvers are very good in solving conjunctive normal form (CNF) formulas. However, some application problems involve lots of parity (xor) constraints which are not necessarily efficiently handled if translated into CNF. This paper studies solving CNF formulas augmented with xor-clauses in the DPLL(XOR) framework where a CDCL SAT solver is coupled with a separate xor-reasoning module. New techniques for analyzing xor-reasoning derivations are developed, allowing one to obtain smaller CNF clausal explanations for xor-implied literals and also to derive and learn new xor-clauses. It is proven that these new techniques allow very short unsatisfiability proofs for some formulas whose CNF translations do not have polynomial size resolution proofs, even when a very simple xor-reasoning module capable only of unit propagation is applied. The efficiency of the proposed techniques is evaluated on a set of challenging logical cryptanalysis instances.

1 Introduction

Modern propositional satisfiability (SAT) solvers (see e.g. [1]) have been successfully applied in a number of industrial application domains. Propositional satisfiability instances are typically encoded in conjunctive normal form (CNF) which allows very efficient Boolean constraint propagation and conflict-driven clause learning (CDCL) techniques. However, such CNF encodings may not allow optimal exploitation of the problem structure in the presence of parity (xor) constraints; such constraints are abundant especially in the logical cryptanalysis domain and also present in circuit verification and bounded model checking. An instance consisting only of parity constraints can be solved in polynomial time using Gaussian elimination, but even state-of-the-art SAT solvers relying only on basic Boolean constraint propagation and CDCL can scale poorly on the corresponding CNF encoding.

In this paper we develop new techniques for exploiting structural properties of xor constraints (i.e. linear equations modulo 2) in the recently introduced DPLL(XOR) framework [2, 3] where a problem instance is given as a combination of CNF and xor-clauses. In the framework a CDCL SAT solver takes care of the CNF part while a separate xor-reasoning module performs propagation on the xor-clauses. In this paper we introduce new techniques for explaining why a literal was implied or why a conflict

* This work has been financially supported by the Academy of Finland under the Finnish Centre of Excellence in Computational Inference (COIN).

occurred in the xor-clauses part; such explanations are needed by the CDCL part. The new core idea is to not see xor-level propagations as implications but as linear arithmetic equations. As a result, the new proposed *parity explanation* techniques can (i) provide smaller clausal explanations for the CDCL part, and also (ii) derive new xor-clauses that can then be learned in the xor-clauses part. The goal of learning new xor-clauses is, similarly to clause learning in CDCL solvers, to enhance the deduction capabilities of the reasoning engine. We introduce the new techniques on a very simple xor-reasoning module allowing only unit propagation on xor-clauses and prove that, even when new xor-clauses are not learned, the resulting system with parity explanations can efficiently solve parity problems whose CNF translations are very hard for resolution. We then show that the new parity explanation techniques also extend to more general xor-reasoning modules, for instance to the one in [2] capable of equivalence reasoning in addition to unit propagation. Finally, we experimentally evaluate the effect of the proposed techniques on a challenging benchmark set modelling cryptographic attacks.

Related work. In [4] a calculus combining basic DPLL without clause learning and Gauss elimination is proposed; their Gauss rules are similar to the general rule $\oplus$-Gen we use in Sect. 8. The solvers EqSatz [5], lsat [6], and March_eq [7] incorporate parity reasoning into DPLL without clause learning, extracting parity constraint information from CNF and during look-ahead, and exploiting it during the preprocessing phase and search. MoRsat [8] extracts parity constraints from a CNF formula, uses them for simplification during preprocessing, and proposes a watched literal scheme for unit propagation on parity constraints. Cryptominisat [9, 10], like our approach, accepts a combination of CNF and xor-clauses as input. It uses the computationally relatively expensive Gaussian elimination as the xor-reasoning method and by default only applies it at the first levels of the search; we apply lighter weight xor-reasoning at all search levels. Standard clause learning is supported in MoRsat and Cryptominisat; our deduction system characterization of xor-reasoning allows us to exploit the linear properties of xor-clauses to obtain smaller CNF explanations of xor-implied literals and xor-conflicts as well as to derive and learn new xor-clauses.

2 Preliminaries

An *atom* is either a propositional variable or the special symbol $\top$ which denotes the constant "true". A *literal* is an atom A or its negation $\neg A$; we identify $\neg\top$ with $\bot$ and $\neg\neg A$ with A. A traditional, non-exclusive *or-clause* is a disjunction $l_1 \vee \cdots \vee l_n$ of literals. An *xor-clause* is an expression of form $l_1 \oplus \cdots \oplus l_n$, where $l_1, \ldots, l_n$ are literals and the symbol $\oplus$ stands for the exclusive logical or. In the rest of the paper, we implicitly assume that each xor-clause is in a *normal form* such that (i) each atom occurs at most once in it, and (ii) all the literals in it are positive. The unique (up to reordering of the atoms) normal form for an xor-clause can be obtained by applying the following rewrite rules in any order until saturation: (i) $\neg A \oplus C \rightsquigarrow A \oplus \top \oplus C$, and (ii) $A \oplus A \oplus C \rightsquigarrow C$, where C is a possibly empty xor-clause and A is an atom. For instance, the normal form of $\neg x_1 \oplus x_2 \oplus x_3 \oplus x_3$ is $x_1 \oplus x_2 \oplus \top$, while the normal form of $x_1 \oplus x_1$ is the empty xor-clause $()$. We say that an xor-clause is *unary* if it is either of form x or $x \oplus \top$ for some variable x; we will identify $x \oplus \top$ with the literal $\neg x$. An

xor-clause is *binary* (*ternary*) if its normal form has two (three) variables. A *clause* is either an or-clause or an xor-clause.

A *truth assignment* π is a set of literals such that $\top \in \pi$ and $\forall l \in \pi : \neg l \notin \pi$. We define the "satisfies" relation $\models$ between a truth assignment π and logical constructs as follows: (i) if l is a literal, then $\pi \models l$ iff $l \in \pi$, (ii) if $C = (l_1 \vee \cdots \vee l_n)$ is an or-clause, then $\pi \models C$ iff $\pi \models l_i$ for some $l_i \in \{l_1, \ldots, l_n\}$, and (iii) if $C = (l_1 \oplus \cdots \oplus l_n)$ is an xor-clause, then $\pi \models C$ iff π is total for C (i.e. $\forall 1 \leq i \leq n : l_i \in \pi \vee \neg l_i \in \pi$) and $\pi \models l_i$ for an odd number of literals of C. Observe that no truth assignment satisfies the empty or-clause $()$ or the empty xor-clause $()$, i.e. these clauses are synonyms for $\bot$.

A *cnf-xor formula* ϕ is a conjunction of clauses, expressible as a conjunction

$$\phi = \phi_{\text{or}} \wedge \phi_{\text{xor}}, \tag{1}$$

where ϕ_{or} is a conjunction of or-clauses and ϕ_{xor} is a conjunction of xor-clauses. A truth assignment π *satisfies* ϕ, denoted by $\pi \models \phi$, if it satisfies each clause in it; ϕ is called *satisfiable* if there exists such a truth assignment satisfying it, and *unsatisfiable* otherwise. The *cnf-xor satisfiability* problem studied in this paper is to decide whether a given cnf-xor formula has a satisfying truth assignment. A formula ϕ' is a *logical consequence* of a formula ϕ, denoted by $\phi \models \phi'$, if $\pi \models \phi$ implies $\pi \models \phi'$ for all truth assignments π. The set of variables occurring in a formula ϕ is denoted by $\text{vars}(\phi)$, and $\text{lits}(\phi) = \{x, \neg x \mid x \in \text{vars}(\phi)\}$ is the set of literals over $\text{vars}(\phi)$. We use $C\,[A/D]$ to denote the (normal form) xor-clause that is identical to C except that all occurrences of the atom A in C are substituted with D once. For instance, $(x_1 \oplus x_2 \oplus x_3)\,[x_1/(x_1 \oplus x_3)] = x_1 \oplus x_3 \oplus x_2 \oplus x_3 = x_1 \oplus x_2$.

3 The DPLL(XOR) framework

The idea in the DPLL(XOR) framework [2] for satisfiability solving of cnf-xor formulas $\phi = \phi_{\text{or}} \wedge \phi_{\text{xor}}$ is similar to that in the DPLL(T) framework for solving satisfiability of quantifier-free first-order formulas modulo a background theory T (SMT, see e.g. [11, 12]). In DPLL(XOR), see Fig. 1 for a high-level pseudo-code, one employs a conflict-driven clause learning (CDCL) SAT solver (see e.g. [1]) to search for a satisfying truth assignment π over all the variables in $\phi = \phi_{\text{or}} \wedge \phi_{\text{xor}}$.[1] The CDCL-part takes care of the usual unit clause propagation on the cnf-part ϕ_{or} of the formula (line 4 in Fig. 1), conflict analysis and non-chronological backtracking (line 15–17), and heuristic selection of decision literals (lines 19–20) which extend the current partial truth assignment π towards a total one.

To handle the parity constraints in the xor-part ϕ_{xor}, an *xor-reasoning module* M is coupled with the CDCL solver. The values assigned in π to the variables in $\text{vars}(\phi_{\text{xor}})$ by the CDCL solver are communicated as *xor-assumption literals* to the module (with the ASSIGN method on line 6 of the pseudo-code). If $l_1, ..., l_m$ are the xor-assumptions communicated to the module so far, then the DEDUCE method (invoked on line 7) of the module is used to deduce a (possibly empty) list of *xor-implied literals* $\hat{l}$ that are logical consequences of the xor-part ϕ_{xor} and xor-assumptions, i.e. literals for which

[1] See [2] for a discussion on handling "xor-internal" variables occurring in ϕ_{xor} but not in ϕ_{or}.

```
solve(φ = φ_or ∧ φ_xor):
1.  initialize xor-reasoning module M with φ_xor
2.  π = ⟨⟩        /*the truth assignment*/
3.  while true:
4.      (π', confl) = UNITPROP(φ_or, π)    /*unit propagation*/
5.      if not confl:        /*apply xor-reasoning*/
6.          for each literal l in π' but not in π: M.ASSIGN(l)
7.          (l̂_1, ..., l̂_k) = M.DEDUCE()
8.          for i = 1 to k:
9.              C = M.EXPLAIN(l̂_i)
10.             if l̂_i = ⊥ or ¬l̂_i ∈ π': confl = C, break
11.             else if l̂_i ∉ π': add l̂_i to π' with the implying or-clause C
12.         if k > 0 and not confl:
13.             π = π'; continue    /*unit propagate further*/
14.     let π = π'
15.     if confl:        /*standard Boolean conflict analysis*/
16.         analyze conflict, learn a conflict clause
17.         backjump or return "unsatisfiable" if not possible
18.     else:
19.         add a heuristically selected unassigned literal in φ to π
20.         or return "satisfiable" if no such variable exists
```

Fig. 1. The essential skeleton of the DPLL(XOR) framework

$\phi_{\text{xor}} \wedge l_1 \wedge ... \wedge l_m \models \hat{l}$ holds. These xor-implied literals can then be added to the current truth assignment π (line 11) and the CDCL part invoked again to perform unit clause propagation on these. The conflict analysis engine of CDCL solvers requires that each implied (i.e. non-decision) literal has an *implying clause*, i.e. an or-clause that forces the value of the literal by unit propagation on the values of literals appearing earlier in the truth assignment (which at the implementation level is a sequence of literals instead of a set). For this purpose the xor-reasoning module has a method EXPLAIN that, for each xor-implied literal $\hat{l}$, gives an or-clause C of form $l'_1 \wedge ... \wedge l'_k \Rightarrow \hat{l}$, i.e. $\neg l'_1 \vee ... \vee \neg l'_k \vee \hat{l}$, such that (i) C is a logical consequence of ϕ_{xor}, and (ii) $l'_1, ..., l'_k$ are xor-assumptions made or xor-implied literals returned before $\hat{l}$. An important special case occurs when the "false" literal $\bot$ is returned as an xor-implied literal (line 10), i.e. when an *xor-conflict* occurs; this implies that $\phi_{\text{xor}} \wedge l_1 \wedge ... \wedge l_m$ is unsatisfiable. In such a case, the clause returned by the EXPLAIN method is used as the unsatisfied clause *confl* initiating the conflict analysis engine of the CDCL part (lines 10 and 15–17).

In addition to the ASSIGN, DEDUCE, and EXPLAIN methods, an xor-reasoning module must also implement methods that allow xor-assumptions to be retracted from the solver in order to allow backtracking in synchronization with the CDCL part (line 17).

Naturally, there are many *xor-module integration strategies* that can be considered in addition to the one described in the above pseudo-code. For instance, the xor-explanations for the xor-implied literals can be computed always (as in the pseudo-code

$$\oplus\text{-Unit}^{+}\colon \frac{x \qquad C}{C\,[x/\top]} \qquad \oplus\text{-Unit}^{-}\colon \frac{x \oplus \top \qquad C}{C\,[x/\bot]}$$

Fig. 2. Inference rules of UP; the symbol x is variable and C is an xor-clause

for the sake of simplicity) or only when needed in the CDCL-part conflict analysis (as in a real implementation for efficiency reasons).

4 The xor-reasoning module "UP"

To illustrate our new parity-based techniques, we first introduce a very simple xor-reasoning module "UP" which only performs unit propagation on xor-clauses. As such it can only perform the same deduction as CNF-level unit propagation would on the CNF translation of the xor-clauses. However, with our new parity-based xor-implied literal explanation techniques (Sect. 5) we can deduce much stronger clauses (Sect. 6) and also new xor-clauses that can be learned (Sect. 7). In Sect. 8 we then generalize the results to other xor-reasoning modules such as the the one in [2] incorporating also equivalence reasoning.

As explained above, given a conjunction of xor-clauses ϕ_{xor} and a sequence $l_1, \ldots, l_k$ of xor-assumption literals, the goal of an xor-reasoning module is to deduce xor-implied literals and xor-conflicts over $\psi = \phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$. To do this, the UP-module implements a deduction system with the inference rules shown in Fig. 2. An UP-*derivation* on ψ is a finite, vertex-labeled directed acyclic graph $G = \langle V, E, L \rangle$, where each vertex $v \in V$ is labeled with an xor-clause $L(v)$ and the following holds for each vertex v:

1. v has no incoming edges (i.e. is an *input vertex*) and $L(v)$ is an xor-clause in ψ, or
2. v has two incoming edges originating from vertices v_1 and v_2, and $L(v)$ is derived from $L(v_1)$ and $L(v_2)$ by using one of the inference rules.

As an example, Fig. 3 shows a UP-derivation for $\phi_{\text{xor}} \wedge (\neg a) \wedge (d) \wedge (\neg b)$, where $\phi_{\text{xor}} = (a \oplus b \oplus c) \wedge (c \oplus d \oplus e) \wedge (c \oplus e \oplus f)$ (please ignore the "cut" lines for now). An xor-clause C is UP-*derivable* on ψ, denoted by $\psi \vdash_{\text{UP}} C$, if there exists a UP-derivation on ψ that contains a vertex labeled with C; the UP-derivable unary xor-clauses are the xor-implied literals that the UP-module returns when its DEDUCE method is called. In Fig. 3, the literal f is UP-derivable and the UP-module returns f as an xor-implied literal after $\neg a$, d, and $\neg b$ are given as xor-assumptions. As a direct consequence of the definition of xor-derivations and the soundness of the inference rules, it holds that if an xor-derivation on ψ contains a vertex labeled with the xor-clause C, then C is a logical consequence of ψ, i.e. $\psi \vdash_{\text{UP}} C$ implies $\psi \models C$. A UP-derivation on ψ is a UP-*refutation of*

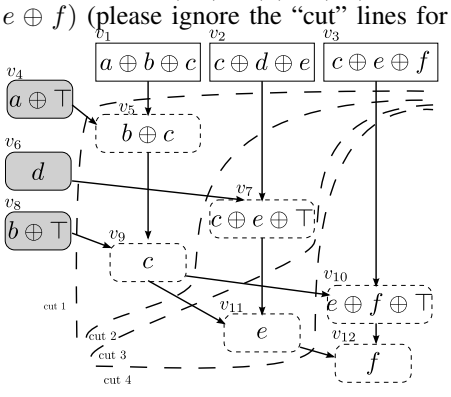

Fig. 3: A UP-derivation

ψ if it contains a vertex labeled with the false literal $\bot$; in this case, ψ is unsatisfiable. A UP-derivation G on ψ is *saturated* if for each unary xor-clause C such that $\psi \vdash_{\mathsf{UP}} C$ it holds that there is a vertex v in G with the label $L(v) = C$. Note that UP is not refutationally complete, e.g. there is no UP-refutation of the unsatisfiable conjunction $(a \oplus b) \wedge (a \oplus b \oplus \top)$. However, it is "eventually refutationally complete" in the DPLL(XOR) setting: if each variable in ψ occurs in a unary clause in ψ, then the empty clause is UP-derivable iff ψ is unsatisfiable; thus when the CDCL SAT solver has assigned a value to all variables in ϕ_{xor}, the UP-module can check whether all the xor-clauses are satisfied.

As explained in the previous section, the CDCL part of the DPLL(XOR) framework requires an implying or-clause for each xor-implied literal. These can be computed by interpreting the $\oplus$-Unit$^+$ and $\oplus$-Unit$^-$ rules as implications

$$(x) \wedge C \Rightarrow C\,[x/\top] \tag{2}$$

$$(x \oplus \top) \wedge C \Rightarrow C\,[x/\bot] \tag{3}$$

respectively, and recursively expanding the xor-implied literal with the left-hand side conjunctions of these until a certain cut of the UP-derivation is reached. Formally, a *cut* of a UP-derivation $G = \langle V, E, L\rangle$ is a partitioning $(V_{\text{a}}, V_{\text{b}})$ of V. A *cut for a non-input vertex* $v \in V$ is a cut $(V_{\text{a}}, V_{\text{b}})$ such that (i) $v \in V_{\text{b}}$, and (ii) if $v' \in V$ is an input vertex and there is a path from v' to v, then $v' \in V_{\text{a}}$. Now assume a UP-derivation $G = \langle V, E, L\rangle$ for $\phi_{\text{xor}} \wedge l_1 \wedge ... \wedge l_k$. For each non-input node v in G, and each cut $W = \langle V_{\text{a}}, V_{\text{b}}\rangle$ of G for v, the *implicative explanation* of v under W is the conjunction $\mathit{Expl}(v, W) = f_W(v)$, there f_W is recursively defined as follows:

E1 If u is an input node with $L(u) \in \phi_{\text{xor}}$, then $f_W(u) = \top$.
E2 If u is an input node with $L(u) \in \{l_1, ..., l_k\}$, then $f_W(u) = L(u)$.
E3 If u is a non-input node in V_{a}, then $f_W(u) = L(u)$.
E4 If u is a non-input node in V_{b}, then $f_W(u) = f_W(u_1) \wedge f_W(u_2)$, where u_1 and u_2 are the source nodes of the two edges incoming to u.

Based on Eqs. (2) and (3), it is easy to see that $\phi_{\text{xor}} \models \mathit{Expl}(v, W) \Rightarrow L(v)$ holds. The implicative explanation $\mathit{Expl}(v, W)$ can in fact be read directly from the cut W as in [2]: $\mathit{Expl}(v, W) = \bigwedge_{u \in \text{reasons}(W)} L(u)$, where $\text{reasons}(W) = \{u \in V_{\text{a}} \mid L(u) \notin \phi_{\text{xor}} \wedge \exists u' \in V_{\text{b}} : \langle u, u'\rangle \in E\}$ is the *reason set* for W. A cut W is *cnf-compatible* if $L(u)$ is a unary xor-clause for each $u \in \text{reasons}(W)$. Thus if the cut W is cnf-compatible, then $\mathit{Expl}(v, W) \Rightarrow L(v)$ is the required or-clause implying the xor-implied literal $L(v)$.

Example 1. Consider again the UP-derivation on $\phi_{\text{xor}} \wedge (\neg a) \wedge (d) \wedge (\neg b)$ in Fig. 3. It has four cuts, 1–4, for the vertex v_{12}, corresponding to the explanations $\neg a \wedge d \wedge \neg b$, $c \wedge d$, $c \wedge (c \oplus e \oplus \top)$, and $e \wedge c$, respectively. The non-cnf-compatible cut 3 cannot be used to give an implying or-clause for the xor-implied literal f but the others can; the one corresponding to the cut 2 is $(\neg c \vee \neg d \vee f)$. ♣

The UP-derivation bears an important similarity with "traditional" implication graph of a SAT solver where each vertex represents a variable assignment: graph partitions are used to derive clausal explanations for implied literals. Different partitioning schemes for such implication graphs have been studied in [13], and we can directly adopt some of them for our analysis. A cut $W = (V_{\text{a}}, V_{\text{b}})$ for a non-input vertex v is:

1. *closest cut* if W is the cnf-compatible cut with the smallest possible V_{b} part. Observe that each implying or-clause derived from these cuts is a clausification of a single xor-clause; e.g., $(\neg c \vee \neg e \vee f)$ obtained from the cut 4 in Fig. 3.
2. *first UIP cut* if W is the cut with the largest possible V_{a} part such that $\mathrm{reasons}(W)$ contains either the latest xor-assumption vertex or exactly one of its successors.
3. *furthest cut* if V_{b} is maximal. Note that furthest cuts are also cnf-compatible as their reason sets consist only of xor-assumptions.

In the implementation of the UP-module, we use a modified version of the 2-watched literals scheme first presented in [14] for or-clauses; all but one of the *variables* in an xor-clause need to be assigned before the xor-clause implies the last one. Thus it suffices to have two *watched variables*. MoRsat [8] uses the same data structure for all clauses and has 2×2 watched literals for xor-clauses. Cryptominisat [9] uses a scheme similar to ours except that it manipulates the polarities of literals in an xor-clause while we take the polarities into account in the explanation phase. Because of this implementation technique, the implementation does not consider the non-unary non-input vertices in UP-derivations; despite this, Thm. 3 does hold also for the implemented inference system.

5 Parity explanations

So far in this paper, as well as in our previous works [2, 3], we have used the inference rules in an "implicative way". For instance, we have implicitly read the $\oplus$-Unit$^+$ rule as

> if the xor-clauses (x) *and* C hold, *then* $C\,[x/\top]$ also holds.

Similarly, the implicative explanation for an xor-implied literal $\hat{l}$ labelling a non-input node v under a cnf-compatible cut W has been defined to be a conjunction $\mathit{Expl}(v, W)$ of literals with $\phi_{\mathrm{xor}} \models \mathit{Expl}(v, W) \Rightarrow \hat{l}$ holding. We now propose an alternative method allowing us to compute a *parity explanation* $\mathit{Expl}_{\oplus}(v, W)$ that is an xor-clause such that

$$\phi_{\mathrm{xor}} \models \mathit{Expl}_{\oplus}(v, W) \Leftrightarrow \hat{l}$$

holds. The variables in $\mathit{Expl}_{\oplus}(v, W)$ will always be a subset of the variables in the implicative explanation $\mathit{Expl}(v, W)$ computed on the same cut.

The key observation for computing parity explanations is that the inference rules can in fact also be read as *equations* over xor-clauses under some provisos. As an example, the $\oplus$-Unit$^+$ rule can be seen as the equation $(x) \oplus C \oplus \top \Leftrightarrow C\,[x/\top]$ *provided that* (i) $x \in C$, and (ii) C is in normal form. That is, taking the exclusive-or of the two premises and the constant true gives us the consequence clause of the rule. The provisos are easy to fulfill: (i) we have already assumed all xor-clauses to be in normal form, and (ii) applying the rule when $x \notin C$ is redundant and can thus be disallowed. The reasoning is analogous for the $\oplus$-Unit$^-$ rule and thus for UP rules we have the equations:

$$(x) \oplus C \oplus \top \Leftrightarrow C\,[x/\top] \tag{4}$$

$$(x \oplus \top) \oplus C \oplus \top \Leftrightarrow C\,[x/\bot] \tag{5}$$

As all the UP-rules can be interpreted as equations of form "left-premise xor right-premise xor true equals consequence", we can expand any xor-clause C in a node of a UP-derivation by iteratively replacing it with the left hand side of the corresponding equation. As a result, we will get an xor-clause that is logically equivalent to C; from this, we can eliminate the xor-clauses in ϕ_{xor} and get an xor-clause D such that $\phi_{\text{xor}} \models D \Leftrightarrow C$. Formally, assume a UP-derivation $G = \langle V, E, L\rangle$ for $\phi_{\text{xor}} \wedge l_1 \wedge ... \wedge l_k$. For each non-input node v in G, and each cut $W = \langle V_{\text{a}}, V_{\text{b}}\rangle$ of G for v, the *parity explanation* of v under W is $\mathit{Expl}_{\oplus}(v, W) = f_W(v)$, there f_W is recursively defined as earlier for $\mathit{Expl}(v, W)$ except that the case "E4" is replaced by

E4 If u is a non-input node in V_{b}, then $f_W(u) = f_W(u_1) \oplus f_W(u_2) \oplus \top$, where u_1 and u_2 are the source nodes of the two edges incoming to u.

We now illustrate parity explanations and show that they can be smaller (in the sense of containing fewer variables) than implicative explanations:

Example 2. Consider again the UP-derivation given in Fig. 3. Take the cut 4 first; we get $\mathit{Expl}_{\oplus}(v_{12}, W) = c \oplus e \oplus \top$. Now $\phi_{\text{xor}} \models \mathit{Expl}_{\oplus}(v_{12}, W) \Leftrightarrow L(v_{12})$ holds as $(c \oplus e \oplus \top) \Leftrightarrow f$, i.e. $c \oplus e \oplus f$, is an xor-clause in ϕ_{xor}. Observe that the implicative explanation $c \wedge e$ of v_{12} under the cut is just one conjunct in the disjunctive normal form $(c \wedge e) \vee (\neg c \wedge \neg e)$ of $c \oplus e \oplus \top$.

On the other hand, under the cut 2 we get $\mathit{Expl}_{\oplus}(v_{12}, W) = d$. Now $\phi_{\text{xor}} \models \mathit{Expl}_{\oplus}(v_{12}, W) \Leftrightarrow L(v_{12})$ as $d \Leftrightarrow f$, i.e. $d \oplus f \oplus \top$, is a linear combination of the xor-clauses in ϕ_{xor}. Note that the implicative explanation for v_{12} under the cut is $(c \wedge d)$, and no cnf-compatible cut for v_{12} gives the implicative explanation (d) for v_{12}. ♣

We observe that $\text{vars}(\mathit{Expl}_{\oplus}(v, W)) \subseteq \text{vars}(\mathit{Expl}(v, W))$ by comparing the definitions of $\mathit{Expl}(v, W)$ and $\mathit{Expl}_{\oplus}(v, W)$. The correctness of $\mathit{Expl}_{\oplus}(v, W)$, formalized in the following theorem, can be established by induction and using Eqs. (4) and (5).

Theorem 1. *Let $G = \langle V, E, L\rangle$ be a UP-derivation on $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$, v a node in it, and $W = \langle V_{\text{a}}, V_{\text{b}}\rangle$ a cut for v. It holds that $\phi_{\text{xor}} \models \mathit{Expl}_{\oplus}(v, W) \Leftrightarrow L(v)$.*

Recall that the CNF-part solver requires an implying or-clause C for each xor-implied literal, forcing the value of the literal by unit propagation. A parity explanation can be used to get such implying or-clause by taking the implicative explanation as a basis and omitting the literals on variables not occurring in the parity explanation:

Theorem 2. *Let $G = \langle V, E, L\rangle$ be a UP-derivation on $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$, v a node with $L(v) = \hat{l}$ in it, and $W = \langle V_{\text{a}}, V_{\text{b}}\rangle$ a cnf-compatible cut for v. Then $\phi_{\text{xor}} \models (\bigwedge_{u \in S} L(u)) \Rightarrow \hat{l}$, where $S = \{u \in \text{reasons}(W) \mid \text{vars}(L(u)) \subseteq \text{vars}(\mathit{Expl}_{\oplus}(v, W))\}$.*

Observing that only expressions of the type $f_W(u)$ occurring an odd number of times in the expression $f_W(v)$ remain in $\mathit{Expl}_{\oplus}(v, W)$, we can derive a more efficient graph traversal method for computing parity explanations. That is, when computing a parity explanation for a node, we traverse the derivation backwards from it in a breadth-first order. If we come to a node u and note that its traversal is requested because an even number of its successors have been traversed, then we don't need to traverse u further or include $L(u)$ in the explanation if u was on the "reason side" V_{a} of the cut.

Example 3. Consider again the UP-derivation in Fig. 3 and the cnf-compatible cut 1 for v_{12}. When we traverse the derivation backwards, we observe that the node v_9 has an even number of traversed successors; we thus don't traverse it (and consequently neither v_8, v_5, v_4 or v_1). On the other hand, v_6 has an odd number of traversed successors and it is included when computing $\mathit{Expl}_{\oplus}(v_{12}, W)$. Thus we get $\mathit{Expl}_{\oplus}(v_{12}, W) = L(v_6) = (d)$ and the implying or-clause for f is $d \Rightarrow f$, i.e. $(\neg d \vee f)$. ♣

Although parity explanations can be computed quite fast using graph traversal as explained above, this can still be computationally prohibitive on "xor-intensive" instances because a single CNF-level conflict analysis may require that implying or-clauses for hundreds of xor-implied literals are computed. In our current implementation, we compute the closest cnf-compatible cut (for which parity explanations are very fast to compute but equal to implicative explanations and produce clausifications of single xor-clauses as implying or-clauses) for an xor-implied literal $\hat{l}$ when an explanation is needed in the regular conflict analysis. The computationally more expensive furthest cut is used if an explanation is needed again in the conflict-clause minimization phase of minisat.

6 Resolution cannot polynomially simulate parity explanations

Intuitively, as parity explanations can contain fewer variables than implicative explanations, the implying or-clauses derived from them should help pruning the remaining search space of the CDCL solver better. We now show that, in theory, parity explanations can indeed be very effective as they can allow small refutations for some formula classes whose CNF translations do not have polynomial size resolution proofs. To do this, we use the hard formulas defined in [15]; these are derived from a class of graphs which we will refer to as "parity graphs". A *parity graph* is an undirected, connected, edge-labeled graph $G = \langle V, E \rangle$ where each node $v \in V$ is labeled with a *charge* $c(v) \in \{\bot, \top\}$ and each edge $\langle v, u \rangle \in E$ is labeled with a distinct variable. The *total charge* $c(G) = \bigoplus_{v \in V} c(v)$ of an parity graph G is the parity of all node charges. Given a node v, define the xor-clause $\alpha(v) = q_1 \oplus \ldots \oplus q_n \oplus c(v) \oplus \top$, where $q_1, \ldots, q_n$ are the variables used as labels in the edges connected to v, and $\mathrm{xorclauses}(G) = \bigwedge_{v \in V} \alpha(v)$. For an xor-clause C over n variables, let $\mathrm{cnf}(C)$ denote the equivalent CNF formula, i.e. the conjunction of 2^{n-1} clauses with n literals in each. Define $\mathrm{clauses}(G) = \bigwedge_{v \in V} \mathrm{cnf}(\alpha(v))$.

As proven in Lemma 4.1 in [15], $\mathrm{xorclauses}(G)$ and $\mathrm{clauses}(G)$ are unsatisfiable if and only if $c(G) = \top$. The unsatisfiable formulas derived from parity graphs can be very hard for resolution: there is an infinite sequence $G_1, G_2, \ldots$ of degree-bounded parity graphs such that $c(G_i) = \top$ for each i and the following holds:

Lemma 1 (Thm. 5.7 of [15]). *There is a constant $c > 1$ such that for sufficiently large m, any resolution refutation of $\mathrm{clauses}(G_m)$ contains c^n distinct clauses, where $\mathrm{clauses}(G_m)$ is of length $\mathcal{O}(n)$, $n = m^2$.*

We now present our key result on parity explanations: for *any* parity graph G with $c(G_i) = \top$, the formula $\mathrm{xorclauses}(G)$ can be refuted with a *single* parity explanation after a number of xor-assumptions have been made:

Theorem 3. *Let $G = \langle V, E\rangle$ be a parity graph such that $c(G) = \top$. There is a* UP*-refutation for* $\mathrm{xorclauses}(G) \wedge q_1 \cdots \wedge q_k$ *for some xor-assumptions $q_1, \ldots, q_k$, a node v with $L(v) = \bot$ in it, and a cut $W = \langle V_\mathrm{a}, V_\mathrm{b}\rangle$ for v such that $Expl_\oplus(v, W) = \top$. Thus* $\mathrm{xorclauses}(G) \models (\top \Leftrightarrow \bot)$*, showing* $\mathrm{xorclauses}(G)$ *unsatisfiable.*

By recalling that CDCL SAT solvers are equally powerful to resolution [16], and that unit propagation on xor-clauses can be efficiently simulated by unit propagation their CNF translation, we get the following:

Corollary 1. *There are families of unsatisfiable cnf-xor formulas for which DPLL(XOR) using* UP*-module (i) has polynomial sized proofs if parity explanations are allowed, but (ii) does not have such if the "classic" implicative explanations are used.*

In practice, the CDCL part does not usually make the correct xor-assumptions needed to compute the empty implying or-clause, but if parity explanations are used in learning as explained in the next section, instances generated from parity graphs can be solved very fast.

7 Learning parity explanations

As explained in Sect. 5, parity explanations can be used to derive implying or-clauses, required by the conflict analysis engine of the CDCL solver, that are shorter than those derived by the classic implicative explanations. In addition to this, parity explanations can be used to derive *new xor-clauses* that are logical consequences of ϕ_xor; these xor-clauses D can then be *learned*, meaning that ϕ_xor is extended to $\phi_\mathrm{xor} \wedge D$, the goal being to increase the deduction power of the xor-reasoning module. As an example, consider again Ex. 2 and recall that the parity explanation for v_{12} under the cut 2 is d. Now $\phi_\mathrm{xor} \models (d \Leftrightarrow f)$, i.e. $\phi_\mathrm{xor} \models (d \oplus f \oplus \top)$, holds, and we can extend ϕ_xor to $\phi'_\mathrm{xor} = \phi_\mathrm{xor} \wedge (d \oplus f \oplus \top)$ while preserving all the satisfying truth assignments. In fact, it is not possible to deduce f from $\phi_\mathrm{xor} \wedge (d)$ by using UP, but f can be deduced from $\phi'_\mathrm{xor} \wedge (d)$. Thus learning new xor-clauses derived from parity explanations can increase the deduction power of the UP inference system in a way similar to conflict-driven clause learning increasing the power of unit propagation in CDCL SAT solvers.

However, if all such derived xor-clauses are learned, it is possible to learn the same xor-clause many times, as illustrated in the following example and Fig. 4.

Example 4. Let $\phi_\mathrm{xor} = (a \oplus b \oplus c \oplus \top) \wedge (b \oplus c \oplus d \oplus e) \wedge \ldots$ and assume that CNF part solver gives its first decision level literals a and $\neg c$ as xor-assumptions to the UP-module; the module deduces b and returns it to the CNF solver. At the next decision level the CNF part guesses d, gives it to UP-module, which deduces e, returns it to the CNF part, and the CNF part propagates it so that a conflict occurs. Now the xor-implied literal e is explained and a new xor-clause $D = (a \oplus d \oplus e \oplus \top)$ is learned in ϕ_xor. After this the CNF part backtracks, implies $\neg d$ at the decision level 1, and gives it to the UP-module; the module can then deduce $\neg e$ *without* using D. If $\neg e$ is now explained, the same "new" xor-clause $(a \oplus d \oplus e \oplus \top)$ can be derived. ♣

The example illustrates a commonly occurring case in which a derived xor-clause contains two or more literals on the latest decision level (e and d in the example); in

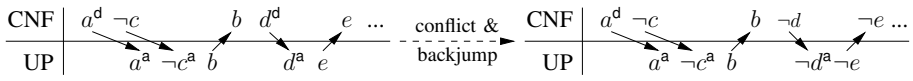

Fig. 4. Communication between CNF part and UP-module in a case when duplicate xor-clauses are learned; the d and a superscripts denote decision literals and xor-assumptions, respectively.

such a case, the xor-clause may already exist in ϕ_{xor}. A conservative approach to avoid learning the same xor-clause twice, under the reasonable assumption that the CNF and xor-reasoning module parts saturate their propagations before new heuristic decisions are made, is to disregard derived xor-clauses that have two or more variables assigned on the latest decision level. If a learned xor-clause for xor-implied literal $\hat{l}$ does not have other literals on the latest decision level, it can be used to infer $\hat{l}$ with fewer decision literals. Note that it may also happen that an implying or-clause for an xor-implied literal $\hat{l}$ does not contain any literals besides $\hat{l}$ on the latest decision level; the CNF part may then compute a conflict clause that does not have any literals on the current decision level, which needs to be treated appropriately.

In order to avoid slowing down propagation in our implementation, we store and remove learned xor-clauses using a strategy adopted from minisat: the maximum number of learned xor-clauses is increased at each restart and the "least active" learned xor-clauses are removed when necessary. However, using the conservative approach to learning xor-clauses, the total number of learned xor-clauses rarely exceeds the number of original xor-clauses.

8 General xor-derivations

So far in this paper we have considered a very simple xor-reasoning module capable only of unit propagation. We can in fact extend the introduced concepts to more general inference systems and derivations. Define an *xor-derivation* similarly to UP-derivation except that there is only one inference rule, $\oplus$-Gen : $\dfrac{C_1 \qquad C_2}{C_1 \oplus C_2 \oplus \top}$, where C_1 and C_2 are xor-clauses. The inference rule $\oplus$-Gen is a generalization of the rules Gauss$^-$ and Gauss$^+$ in [4]. Now Thms. 1 and 2 can be shown to hold for such derivations as well.

As another concrete example of xor-reasoning module implementing a sub-class of $\oplus$-Gen, consider the Subst module presented in [2]. In addition to the unit propagation rules of UP in Fig. 2, it has inference rules allowing equivalence reasoning:

$$\oplus\text{-Eqv}^+ : \frac{x \oplus y \oplus \top \qquad C}{C\,[x/y]} \qquad\qquad \oplus\text{-Eqv}^- : \frac{x \oplus y \qquad C}{C\,[x/(y \oplus \top)]}$$

where the symbols x and y are variables while C is an xor-clause in the normal form with an occurrence of x. Note that these Subst rules are indeed instances of the more general inference rule $\oplus$-Gen. For instance, given two xor-clauses $C_1 = (c \oplus d \oplus \top)$ and $C_2 = (b \oplus d \oplus e)$, the Subst-system can produce the xor-clause $C_2\,[d/c] = (b \oplus c \oplus e)$ which is also inferred by $\oplus$-Gen: $(C_1 \oplus C_2 \oplus \top) = ((c \oplus d \oplus \top) \oplus (b \oplus d \oplus e) \oplus \top) = (b \oplus c \oplus e)$.

Subst-derivations are defined similarly to UP-derivations. As an example, Fig. 5 shows a Subst-derivation on $\phi_{\text{xor}} \wedge (a)$, where $\phi_{\text{xor}} = (a \oplus b \oplus c) \wedge (a \oplus c \oplus d) \wedge (b \oplus d \oplus e)$. The literal e is Subst-derivable on $\phi_{\text{xor}} \wedge (a)$; the xor-reasoning module returns e as an xor-implied literal on ϕ_{xor} after a is given as an xor-assumption. The cnf-compatible cut 1 for the literal e gives the implicative explanation (a) and thus the implying or-clause $(\neg a \vee e)$ for e. Parity explanations are defined for Subst in the same way as for UP; the parity explanation for the literal e in the figure is $\top$ and thus the implying or-clause for e is (e). Observe that e is *not* UP-derivable from $\phi_{\text{xor}} \wedge (a)$, i.e. Subst is a stronger deduction system than UP in this sense.

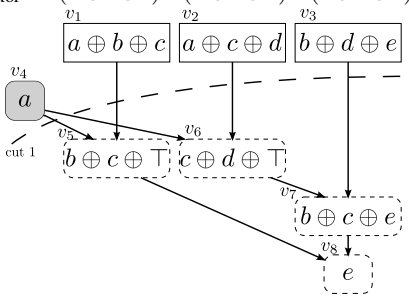

Fig. 5: A Subst-derivation

Parity explanations can also be computed in another xor-reasoning module, EC presented in [3], that is based on equivalence class manipulation. We omit this construction due to space constraints.

9 Experimental results

We have implemented a prototype solver integrating both xor-reasoning modules (UP and Subst) to minisat [17] (version 2.0 core) solver. In the experiments we focus on the domain of logical cryptanalysis by modeling a "known cipher stream" attack on stream ciphers Bivium, Crypto-1, Grain, Hitag2, and Trivium. To evaluate the performance of the proposed techniques, we include both unsatisfiable and satisfiable instances. In the unsatisfiable instances, generated with grain-of-salt [18], the task is to recover the internal cipher state when 256 output stream bits are given. This is infeasible in practice, so the instances are made easier and also unsatisfiable by assigning a large enough number of internal state bits randomly. Thus, the task becomes to prove that it is not possible to assign the remaining bits of the internal cipher state so that the output would match the given bits. To include also satisfiable instances we modeled a different kind of attack on the ciphers Grain, Hitag2 and Trivium where the task is to recover the full key when a small number of cipher stream bits are given. In the attack, the IV and a number of key stream bits are given. There are far fewer generated cipher stream bits than key bits, so a number of keys probably produce the same prefix of the cipher stream. All instances were converted into (i) the standard DIMACS CNF format, and (ii) a DIMACS-like format allowing xor-clauses as well. Structurally these instances are interesting for benchmarking xor-reasoning as they have a large number of tightly connected xor-clauses combined with a significant CNF part.

We first compare the following solver configurations: (i) unmodified minisat, (ii) up: minisat with watched variable based unit propagation on xor-clauses, (iii) up-pexp: up extended with parity explanations, (iv) up-pexp-learn: up-pexp extended with xor-clause learning, and (v) up-subst-learn: up using Subst-module to compute parity xor-explanations and xor-clause learning. The reference configuration up computes closest cnf-compatible cut parity explanations, and the other configurations use also furthest

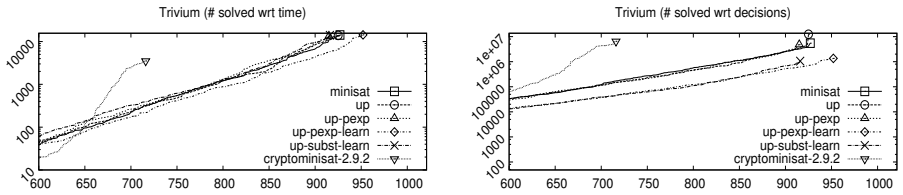

Fig. 6. Number of solved instances with regard to time and decisions on satisfiable Trivium benchmark (1020 instances, 51 instances per generated cipher stream length ranging from 1 to 20 bits)

	Bivium			Crypto-1			Grain			Hitag2			Trivium		
Solver	#	Dec.	Time	#	Dec.	Time	#	Dec.	Time	#	Dec.	Time	#	Dec.	Time
minisat	**51**	834	**80.9**	**51**	781	691.1	1	-	-	35	428	440.0	**51**	55	5.7
up	**51**	985	127.7	**51**	1488	1751.8	**51**	40	13.8	39	291	403.9	**51**	59	8.0
up-pexp	**51**	1040	147.8	**51**	1487	1748.2	**51**	**35**	10.9	37	124	148.0	**51**	62	9.8
up-pexp-learn	47	651	114.0	**51**	1215	1563.0	36	122	87.7	37	222	255.4	**51**	**24**	**3.7**
up-subst-learn	47	616	336.4	**51**	1037	2329.5	37	70	90.3	36	215	374.8	**51**	29	12.9
cryptominisat-2.9.2	**51**	**588**	89.8	**51**	**0**	**0.06**	**51**	89	**10.4**	**51**	**0**	**0.07**	**51**	71	6.04

Fig. 7. Results of the unsatisfiable benchmarks showing the number of solved instances (#) within the 4h time limit, median of decisions ($\times 10^3$), and median of solving time

cuts selectively as described in Sect. 5. We also tested first UIP cuts, but the performance did not improve.

The results for the satisfiable Trivium benchmarks are shown in Fig. 6. Learning xor-clauses reduces the number of decisions needed substantially, and in the case of the computationally less expensive UP reasoning module this is also reflected in the solving time and in the number of solved instances. On the other satisfiable benchmark sets learning new xor-clauses also reduced the number of required decisions significantly but the number of propagations per decision is also greatly increased due to increased deduction power and the reduction is not really reflected in the solving time.

The results for the unsatisfiable benchmarks are shown in Fig. 7. Parity explanations reduce decisions on Grain and Hitag2, leading to fastest solving time. Learning parity explanations reduces explanations on all benchmarks except Grain and gives the best solving time on Trivium. Equivalence reasoning seems to reduce decisions slightly with the cost of increased solving time. Obviously more work has to be done to improve data structures and adjust heuristics so that the theoretical power of parity explanations and xor-clause learning can be fully seen also in practice.

We also ran cryptominisat version 2.9.2 [10] on the benchmarks. As shown in Figs. 6 and 7, it performs (i) extremely well on the unsatisfiable Crypto-1 and Hitag2 instances due to "failed literal detection" and other techniques, but (ii) not so well on our satisfiable Trivium instances, probably due to differences in restart policies or other heuristics.

10 Conclusions

We have shown how to compute linearity exploiting parity explanations for literals deduced in an xor-reasoning module. Such explanations can be used (i) to produce more

compact clausal explanations for the conflict analysis engine of a CDCL solver incorporating the xor-reasoning module, and (ii) to derive new parity constraints that can be learned in order to boost the deduction power of the xor-reasoning module. It has been proven that parity explanations allow very short refutations of some formulas whose CNF translations do not have polynomial size resolution proofs, even when using a simple xor-reasoning module capable only of unit-propagation. The experimental evaluation suggests that parity explanations and xor-clause learning can be efficiently implemented and demonstrates promising performance improvements also in practice.

References

1. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Handbook of Satisfiability. IOS Press (2009)
2. Laitinen, T., Junttila, T., Niemelä, I.: Extending clause learning DPLL with parity reasoning. In: Proc. ECAI 2010, IOS Press (2010) 21–26
3. Laitinen, T., Junttila, T., Niemelä, I.: Equivalence class based parity reasoning with DPLL(XOR). In: Proc. ICTAI 2011, IEEE (2011)
4. Baumgartner, P., Massacci, F.: The taming of the (X)OR. In: Proc. CL 2000. Volume 1861 of LNCS., Springer (2000) 508–522
5. Li, C.M.: Integrating equivalency reasoning into Davis-Putnam procedure. In: Proc. AAAI/IAAI 2000, AAAI Press (2000) 291–296
6. Ostrowski, R., Grégoire, É., Mazure, B., Sais, L.: Recovering and exploiting structural knowledge from CNF formulas. In: Proc. CP 2002. (2002) 185–199
7. Heule, M., Dufour, M., van Zwieten, J., van Maaren, H.: March_eq: Implementing additional reasoning into an efficient look-ahead SAT solver. In: Proc. SAT 2004. Volume 3542 of LNCS., Springer (2004) 345–359
8. Chen, J.: Building a hybrid SAT solver via conflict-driven, look-ahead and XOR reasoning techniques. In: Proc. SAT 2009. Volume 5584 of LNCS., Springer (2009) 298–311
9. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Proc. SAT 2009. Volume 5584 of LNCS., Springer (2009) 244–257
10. Soos, M.: Enhanced gaussian elimination in DPLL-based SAT solvers. In: Pragmatics of SAT, Edinburgh, Scotland, GB (2010) 1–1
11. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). Journal of the ACM **53**(6) (2006) 937–977
12. Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability. IOS Press (2009)
13. Zhang, L., Madigan, C.F., Moskewicz, M.W., Malik, S.: Efficient conflict driven learning in boolean satisfiability solver. In: Proc. ICCAD 2001, IEEE Press (2001) 279–285
14. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proc. DAC 2001, ACM (2001) 530–535
15. Urquhart, A.: Hard examples for resolution. J. ACM **34**(1) (1987) 209–219
16. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers as resolution engines. Artificial Intelligence **175**(2) (2011) 512–525
17. Eén, N., Sörensson, N.: An extensible SAT solver. In: Proc. SAT 2003. Volume 2919 of LNCS., Springer (2004) 502–518
18. Soos, M.: Grain of salt — an automated way to test stream ciphers through SAT solvers. In: Tools'10: Proceedings of the Workshop on Tools for Cryptanalysis 2010, RHUL (2010) 1–2

A Proofs

Theorem 1. *Let $G = \langle V, E, L\rangle$ be a* UP*-derivation on $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$, v a node in it, and $W = \langle V_\text{a}, V_\text{b}\rangle$ a cut for v. It holds that $\phi_{\text{xor}} \models \mathit{Expl}_\oplus(v, W) \Leftrightarrow L(v)$.*

Proof. We show that $\phi_{\text{xor}} \models (f_W(u) \Leftrightarrow L(u))$ for each node u in G by induction on the structure of the derivation G.

If u is an input node with $L(u) \in \phi_{\text{xor}}$, then $f_W(u) = \top$ and $\phi_{\text{xor}} \models (\top \Leftrightarrow L(u))$ as $L(u)$ is a conjunct in ϕ_{xor}.

If u is an input node with $L(u) \in \{l_1, ..., l_k\}$, then $f_W(u) = L(u)$ and $\phi_{\text{xor}} \models (L(u) \Leftrightarrow L(u))$ holds trivially.

If u is a non-input node in V_a, then $f_W(u) = L(u)$ and $\phi_{\text{xor}} \models (L(u) \Leftrightarrow L(u))$ holds trivially.

If u is a non-input node in V_b with two incoming edges from nodes u_1 and u_2 respectively, then $f_W(u) = f_W(u_1) \oplus f_W(u_2) \oplus \top$. Because $L(u)$ is obtained from $L(u_1)$ and $L(u_2)$ by applying either $\oplus$-Unit$^+$ or $\oplus$-Unit$^-$, and the Eqs. (4) and (5) are valid, we have $\phi_{\text{xor}} \models (L(u_1) \oplus L(u_2) \oplus \top \Leftrightarrow L(u))$. By the induction hypothesis the equations $\phi_{\text{xor}} \models (f_W(u_1) \Leftrightarrow L(u_1))$ and $\phi_{\text{xor}} \models (f_W(u_2) \Leftrightarrow L(u_2))$ hold, so we have $\phi_{\text{xor}} \models (f_W(u_1) \oplus f_W(u_2) \oplus \top \Leftrightarrow L(u))$, i.e. $\phi_{\text{xor}} \models (f_W(u) \Leftrightarrow L(u))$.

For each u in G, it holds that $\phi_{\text{xor}} \models (f_W(u) \Leftrightarrow L(u))$. It follows that $\phi_{\text{xor}} \models \mathit{Expl}_\oplus(v, W) \Leftrightarrow L(v)$. □

Theorem 2. *Let $G = \langle V, E, L\rangle$ be a* UP*-derivation on $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$, v a node with $L(v) = \hat{l}$ in it, and $W = \langle V_\text{a}, V_\text{b}\rangle$ a cnf-compatible cut for v. Then $\phi_{\text{xor}} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \hat{l}$, where $S = \{u \in \text{reasons}(W) \mid \text{vars}(L(u)) \subseteq \text{vars}(\mathit{Expl}_\oplus(v, W))\}$.*

Proof. If ϕ_{xor} is unsatisfiable, then $\phi_{\text{xor}} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \hat{l}$ holds trivially.

Assume that ϕ_{xor} is satisfiable. As $\text{vars}(\bigwedge_{u\in S} L(u)) = \text{vars}(\mathit{Expl}_\oplus(v, W))$ and $\phi_{\text{xor}} \models \mathit{Expl}_\oplus(v, W) \Leftrightarrow \hat{l}$ holds, it must be that either $\phi_{\text{xor}} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \hat{l}$ or $\phi_{\text{xor}} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \neg\hat{l}$ holds (both cannot hold because then ϕ_{xor} would be unsatisfiable). If $\phi_{\text{xor}} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \neg\hat{l}$ holds, then, as $S \subseteq \text{reasons}(W)$ holds, $\phi_{\text{xor}} \models (\bigwedge_{u\in\text{reasons}(W)} L(u)) \Rightarrow \neg\hat{l}$ would also hold. Combined with the property $\phi_{\text{xor}} \models (\bigwedge_{u\in\text{reasons}(W)} L(u)) \Rightarrow \hat{l}$ of implicative explanations, this means that ϕ_{xor} is unsatisfiable, contradicting our assumption. Therefore, $\phi_{\text{xor}} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \hat{l}$ must hold. □

Theorem 3. *Let $G = \langle V, E\rangle$ be a parity graph such that $c(G) = \top$. There is a* UP*-refutation for* $\text{xorclauses}(G) \wedge q_1 \cdots \wedge q_k$ *for some xor-assumptions $q_1, \ldots, q_k$, a node v with $L(v) = \bot$ in it, and a cut $W = \langle V_\text{a}, V_\text{b}\rangle$ for v such that $\mathit{Expl}_\oplus(v, W) = \top$. Thus* $\text{xorclauses}(G) \models (\top \Leftrightarrow \bot)$*, showing* $\text{xorclauses}(G)$ *unsatisfiable.*

Proof. Let $E' \subseteq E$ be a spanning tree of G, and $q_1, \ldots, q_k$ the variables occurring on the labels of the edges in $E \backslash E'$. We construct a UP-refutation G' for $\text{clauses}(G) \wedge q_1 \wedge \cdots \wedge q_k$ by applying $\oplus$-Unit$^+$ twice for each variable q_i. Note for every leaf node u in the spanning tree it holds that $C = \alpha(u)\,[q_1/\top] \ldots [q_k/\top]$ is a unit xor-clause C fixing the value of some variable x. We pick a leaf node u, propagate value of the variable

x by applying the rule $\oplus$-Unit$^+$ or $\oplus$-Unit$^-$, and remove the node u from the spanning tree. We proceed until all nodes in the spanning tree are eliminated and we have derived an empty xor-clause $L(v) = \bot$ for a node v in the UP-refutation G'. Take the furthest cut W. Computing the parity explanation for v we will get $Expl_{\oplus}(v, W) = \top$ as there are exactly two paths from each xor-assumption node q_i to v and an odd number of non-input nodes. The construction is illustrated in Fig. 8. □

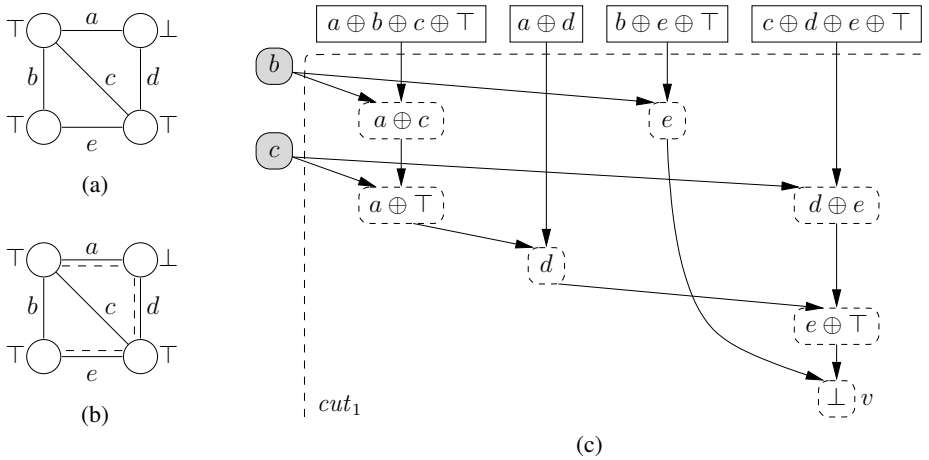

Fig. 8. (a) a parity graph G, (b) a spanning tree of the graph, and (c) a UP-refutation for $\text{xorclauses}(G) \wedge b \wedge c$ with $Expl_{\oplus}(v, cut_1) = \top$.