BOSPHORUS: Bridging ANF and CNF Solvers

Davin Choo*, Mate Soos†, Kian Ming A. Chai*, and Kuldeep S. Meel†

*Information Division, DSO National Laboratories, Singapore
† School of Computing, National University of Singapore

***Abstract*—Algebraic Normal Form (ANF) and Conjunctive Normal Form (CNF) are commonly used to encode problems in Boolean algebra. ANFs are typically solved via Gröbner basis algorithms, often using more memory than is feasible; while CNFs are solved using SAT solvers, which cannot exploit the algebra of polynomials naturally. We propose a paradigm that bridges between ANF and CNF solving techniques: the techniques are applied in an iterative manner to *learn facts* to augment the original problems. Experiments on over 1,100 benchmarks arising from four different applications domains demonstrate that learnt facts can significantly improve runtime and enable more benchmarks to be solved.**

I. INTRODUCTION

Algebraic Normal Form (ANF) and Conjunctive Normal Form (CNF) are two commonly used normal forms in Boolean algebra. Both ANF and CNF reason about Boolean variables $x_1, \ldots, x_n$ but with different Boolean operators.

ANF is a *system of polynomial equations* in GF(2), i.e., the Galois field of two elements, or $\mathbb{Z}_2$. Each polynomial is a sum of monomials, where a monomial is a product of zero or more variables. Cryptologists prefer ANF because it naturally encodes definitions such as AES [1] and hash functions [2].

One approach to solving ANF is to compute the Gröbner basis of the system using the Buchberger's algorithm [3] or its variants [4], [5]. Efficient implementations include M4GB [6], FGb [7] and Magma [8]. In certain systems, methods such as XL/XSL [9], [10] and ElimLin [11], [12] have also been shown to be effective. Unfortunately, ANF solvers on huge polynomial systems tend to require more memory than is feasible on most computing platforms [13].

In comparison, CNF is a *conjunction of clauses*. Each clause is a disjunction of literals, where a literal is either a Boolean variable or its negation. As Boolean circuits are naturally described in logical connectives, hardware verification problems are often described in CNFs [14]. Some other domains using CNFs are software verification, industrial planning, scheduling and recreational mathematical puzzle solving.

CNFs are typically solved by SAT solvers, which use significantly less memory than the methods for ANF. This is primarily due to the depth-first search nature of CDCL [15] that most modern SAT solvers are based on. Many solvers build upon the small code base of MiniSat [16], which includes the standard CDCL, variable and clause elimination [17], watched literals data structures [18] and the like.

The open-source tool is available at https://github.com/meelgroup/bosphorus

ANF and CNF solving algorithms exploit different properties of the problem encoding. For instance, Gauss-Jordan elimination (GJE) is a natural procedure in ANF, but not in CNF; while conflict learning prunes the search tree in SAT solvers, but we are unaware of such learning for ANF. Despite the recent successes of GJE-enabled SAT solvers in counting problems [19], [20], the use of GJE-enabled solvers is not prevalent. In this context, we ask: *is there an alternative and easier way to combine ANF and CNF solving?*

The primary contribution of this paper is an affirmative answer to the above question. We demonstrate a paradigm that bridges between ANF and CNF solving techniques. The techniques are applied in an iterative manner to *learn facts* to augment the original problems. This approach is attractive when the conversion time between ANF and CNF encodings is negligible relative to the overall solving time. Our experiments demonstrate that our iterative approach can help us to solve more instances while spending less time.

As a consequence of this bridge, problems can be encoded in their most natural and comprehensible manner, either in ANF or CNF, and yet draws from solving techniques in both to achieve reasonable solving performance — this is our second contribution. We call our tool BOSPHORUS, the namesake of *the Bosphorus bridge* connecting Europe and Asia.

In the next section, we describe the various techniques for solving ANFs and CNFs. Section III describes how BOSPHORUS uses these techniques. Results on three classes of ANF problems and the SAT Competition 2017 benchmarks are in section IV. For notation, we use $\oplus$ for exclusive-OR (XOR) and addition in GF(2), $\neg$ for negation, $\wedge$ for conjunction and $\vee$ for disjunction. We use the term *polynomial* to mean *polynomial equation equated to zero*, and we will also write such equations by just stating the polynomial.

II. LEARNING FACTS

Our approach iteratively extracts two types of *learnt facts*: (1) linear equations $x_{i_1} + x_{i_2} + \cdots + x_{i_p} + c$ where c is either zero or one; and (2) polynomials of the form $x_{i_1} x_{i_2} \ldots x_{i_p} \oplus 1$. The former keeps the degree of the system low while the latter allows immediate deduction that $x_{i_1} = x_{i_2} = \cdots = x_{i_p} = 1$. The rest of this section explains how BOSPHORUS obtains and uses these facts in various phases.

A. ANF propagation

For each variable, we attempt to assign a *value* (0 or 1) or an *equivalent literal* by examining the polynomials involving the variable. A value assignment can occur in two cases. First, for polynomial x or $x \oplus 1$, we set x to the constants 0 or 1 respectively. Second, for polynomial $x_{i_1} x_{i_2} \dots x_{i_p} \oplus 1$, we set $x_{i_1} = x_{i_2} = \dots = x_{i_p} = 1$. An equivalence assignment happens if the polynomial is $x \oplus y$ or $x \oplus y \oplus 1$, in which case we set $x = y$ or $x = \neg y$ respectively. These assignments are applied iteratively until a fixed point is reached.

B. eXtended Linearization (XL)

Gauss-Jordan elimination (GJE) solves a system of linear equations by elementary row operations. For polynomials, one can apply GJE by treating each monomial as an independent variable — this is known as *linearization*. Dependence between the monomials can be re-introduced by generating more polynomial equations, a process known as eXtended Linearization (XL) [9]. We describe XL and how it is used.

Given a polynomial system S with n variables and m equations, we expand S incrementally to obtain an expanded system S'. The expansion process selects each equation in S in ascending degree order and multiplies the equation with all possible monomials up to a chosen degree D. In the case where we manage to expand S fully, the expanded system will have $m \sum_{j=0}^{D} \binom{n}{j}$ polynomials. GJE is then applied on S'.

Table I shows an example of applying XL on the ANF $\{x_1x_2 \oplus x_1 \oplus 1, x_2x_3 \oplus x_3\}$, expanding up to degree $D = 1$ monomials. The last three rows of Table Ib are the facts $\{x_1 \oplus 1, x_2, x_3\}$ that BOSPHORUS will retain.

Applying XL on the entire ANF often requires considerable memory and time. To avoid this, we uniformly subsample the polynomials from the ANF to obtain an m'-by-n' linearized system S such that $m'n' \gtrsim 2^M$, for a fixed parameter M. Moreover, S is incrementally expanded only until the system size is approximately $2^{M+\delta M}$, for a parameter δM.

We employ XL in this manner because our primary purpose is not to solve the system but to learn facts to augment it. We also employ ElimLin and SAT solver in the same spirit.

C. ElimLin

ElimLin [11] is an algorithm that iterates through the following three steps until fixed point: (1) apply GJE on the linearization of the polynomial system S; (2) gather linear equations and remove them from S, yielding S'; and (3) for each linear equation ℓ, pick, say, a variable from ℓ that occurs in the least number equations in S', and eliminate that variable from S' using ℓ. The resultant system S'' is free of linear equations. The process is repeated from step (1) using S'' as S until there are no more linear equations after applying GJE.

Consider the ANF $\{x_1 \oplus x_2 \oplus x_3, x_1x_2 \oplus x_2x_3 \oplus 1\}$. As step (1) does not affect the system, $x_1 \oplus x_2 \oplus x_3$ remains the only linear equation in step (2). If we choose to substitute x_1 by $x_2 \oplus x_3$ in step (3), the ANF becomes the single equation $(x_2 \oplus x_3)x_2 \oplus x_2x_3 \oplus 1$. By right-distributing the first conjunction over the first XOR and then replacing the XOR of x_2x_3 with itself by zero, this equation simplifies to $x_2 \oplus 1$. Assigning $x_2 = 1$ and performing ANF propagation on the original ANF, $x_1x_2 \oplus x_2x_3 \oplus 1$ becomes $x_1 \oplus x_3 \oplus 1$, and the ANF propagation can deduce the equivalence $x_1 = \neg x_3$.

Similar to XL, we apply ElimLin on a random subset of polynomials that has linearized size of approximately 2^M.

D. Conflict-bounded SAT solving

With a CNF equivalent of the ANF, we call a SAT solver that has conflict-driven clause learning [15]. The solver is allowed up to a pre-determined number C of conflicts to solve the system. We bound the solver using use a conflict budget instead of a time budget for replicability of experiments.

Due to this budget, the solver will surely terminate with one of these three cases: (1) unsatisfiable; (2) satisfiable, giving an assignment; or (3) undecidable within the limit. In case (1), BOSPHORUS appends the contradictory equation $1 = 0$ to the system — this is the learnt fact by the SAT solver. In cases (2) and (3), BOSPHORUS extracts linear equations from learnt clauses — of particular interest are linear equations from the unit and binary clauses because they immediately yield value and equivalence assignments.

E. Example

Consider the ANF

$$\begin{aligned} &x_1x_2 \oplus x_3 \oplus x_4 \oplus 1, \quad && x_1x_2x_3 \oplus x_1 \oplus x_3 \oplus 1, \\ &x_1x_3 \oplus x_3x_4x_5 \oplus x_3, \quad && x_2x_3 \oplus x_3x_5 \oplus 1, \\ &x_2x_3 \oplus x_5 \oplus 1. \end{aligned} \tag{1}$$

XL with $D = 1$ on this system learns the facts $x_2x_3x_4 \oplus 1$, $x_1x_3x_4 \oplus 1$, $x_1 \oplus x_5 \oplus 1$, $x_1 \oplus x_4$, $x_3 \oplus 1$, and $x_1 \oplus x_2$. For ElimLin, its initial GJE — step (1) in section II-C — gives four distinct linear equations: $x_1 \oplus x_5 \oplus 1$; $x_1 \oplus x_4$; $x_3 \oplus 1$; and $x_1 \oplus x_2$. After substituting x_5 by $x_1 \oplus 1$, x_4 by x_1, x_3 by 1 and x_2 by x_1, ElimLin learns $x_1 \oplus 1$. Converting to CNF using Karnaugh map (section III-C) creates one auxiliary variable for x_1x_2. Boolean constraint propagation in the SAT solver then gives $x_2 \oplus 1$, $x_4 \oplus 1$, x_5, and $x_1x_2 \oplus 1$.

ANF propagation using the above facts obtained from XL, ElimLin and SAT solver simplifies the system into

$$x_1 \oplus 1, \qquad x_2 \oplus 1, \qquad x_3 \oplus 1, \qquad x_4 \oplus 1, \qquad x_5. \tag{2}$$

This effectively solves the system to its unique satisfying assignment $x_1 = x_2 = x_3 = x_4 = 1$ and $x_5 = 0$.

Observe that ANF propagation after the XL step would have led to (2) without the need for either ElimLin or SAT solver. Nevertheless, the above example illustrates that each can derive different learnt facts: XL gives the value assignment for x_3, ElimLin gives that for x_1, and the SAT solver learns the remaining assignments. To make full use of these different learnt facts, BOSPHORUS is designed to perform ANF propagation when learnt facts are produced after every step.

III. BOSPHORUS

This section details the workflow and the data structures of BOSPHORUS, and the approaches to convert between ANFs and CNFs. The source code is available at https://github.com/meelgroup/bosphorus.

TABLE I: An example of applying eXtended Linearization (XL). Zero coefficients in the matrices are suppressed; and rows corresponding to zero polynomials are omitted. The last three rows of (b) are the facts that will be retained.

(a) Expansion by degree 1 monomials

		Expanded linearized system							
Polynomial	Multiplier	$x_1x_2x_3$	x_2x_3	x_1x_3	x_1x_2	x_3	x_2	x_1	1
$x_1x_2 \oplus x_1 \oplus 1$	1				1			1	1
	x_1				1				
	x_2						1		
	x_3	1		1		1			
$x_2x_3 \oplus x_3$	1		1			1			
	x_1	1		1					
	x_3		1			1			

(b) Gauss-Jordan Elimination

Linearized system after GJE							
$x_1x_2x_3$	x_2x_3	x_1x_3	x_1x_2	x_3	x_2	x_1	1
1		1					
	1						
			1				
				1			
					1		
						1	**1**

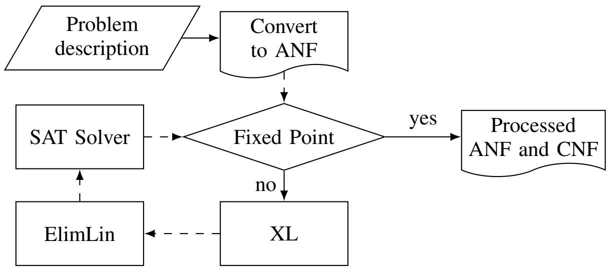

Fig. 1: BOSPHORUS's flow. A dashed arrow means ANF propagation is applied.

A. Workflow

BOSPHORUS takes a problem encoded in ANF and produces a processed ANF and CNF after performing an XL–ElimLin–SAT-solver *fact-learning loop* until the fixed point when no further learnt facts are produced. ANF propagation is performed on the input ANF and whenever learnt facts are produced. Fig. 1 shows the overall workflow.

Internally within BOSPHORUS, the problem is represented as an ANF polynomial system, and only ANF propagation modifies and replaces this master copy. Each of the other techniques — XL, ElimLin and SAT solver — operates on a copy of the ANF, and learnt facts are extracted and then added onto the master copy if not already there.

If the equation $1 = 0$ is detected, BOSPHORUS terminates and returns UNSAT. If the SAT solver finds a satisfying solution, BOSPHORUS stores the solution. This solution is not used to simplify the ANF because it may not be unique.

B. Data structures

BOSPHORUS stores the system of equations in the ANF description as a list of Boolean polynomials. For each variable, we track (i) its value, as either 0, 1, or undetermined; (ii) its equivalence literal; and (iii) its occurrence list.

The default equivalence literal for each variable is the variable itself and may change as BOSPHORUS proceeds. For example, the equivalence literal of x_i may be switched to $\neg x_j$ to encode $x_i = \neg x_j$.

Occurrence list is an optimization technique from the SAT literature [18], [21]. Here, BOSPHORUS tracks the list of polynomials that each variable occurs in. For example, updates to x_1 in (1) do not involve processing the last two equations. The time saved can be significant for large polynomial systems.

C. ANF to CNF conversion

CNF is used by the SAT solver within BOSPHORUS, and it is also an output. To convert ANF to CNF, we introduce an auxiliary CNF variable on-the-fly for each ANF monomial, and we maintain a bi-directional map for such variables.

BOSPHORUS handles determined variables, equivalences, and polynomials differently in the conversion. Determined variables are added as unit clauses, while an equivalence such as $x_i = \neg x_j$ is represented in CNF by $(x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$. For a polynomial, it is first re-expressed as shorter ones by introducing auxiliary variables. The number of terms in the shorter polynomials is parameterized by an XOR-cutting length L, Then, each of these shorter polynomials is converted to CNF using either of the following two approaches:

1) If the polynomial is K-variate, we use the Karnaugh map to yield the minimal clause representation while reducing the number of auxiliary variables used. Because computing the Karnaugh map scales exponentially with the number of variables, the Karnaugh parameter K is kept low to ensure reasonable conversion time.
2) If the polynomial involves more than K variables, we apply a transformation *à la* Tseitin encoding [22]. Each polynomial of length $l \leq L$ is treated as an XOR clause of independent terms and converted to CNF clauses by enumerating through all possible 2^l terms.

Although the Karnaugh map approach is less flexible, it can yield a more compact conversion than the Tseitin-based approach. Consider the polynomial equation $x_1x_3 \oplus x_1 \oplus x_2 \oplus x_4 \oplus 1 = 0$. Fig. 2 shows possible CNF representations via both approaches. Using the Karnaugh map shown in Fig. 3, one can derive a more compact CNF system that directly deals with the variables involved. In comparison, the Tseitin-based approach creates a new CNF variable x_5 and encode $x_5 = x_1x_3$ using three CNF clauses.

At present, any auxiliary variable introduced in the conversion process does not participate in the learnt facts.

D. CNF to ANF conversion

BOSPHORUS can be used as a CNF preprocessor, though its main use-case is that of solving problems represented in ANF.

$$\begin{array}{r}
x_1 \vee x_2 \vee x_4 \\
\neg x_1 \vee \neg x_2 \vee x_3 \vee x_4 \\
x_2 \vee \neg x_3 \vee x_4 \\
\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4 \\
x_1 \vee \neg x_2 \vee \neg x_4 \\
\neg x_2 \vee \neg x_3 \vee \neg x_4
\end{array}
\qquad
\begin{array}{r}
x_1 \vee \neg x_5 \\
x_3 \vee \neg x_5 \\
\neg x_1 \vee \neg x_3 \vee x_5 \\
x_1 \vee x_2 \vee x_4 \vee x_5 \\
\neg x_1 \vee \neg x_2 \vee x_4 \vee x_5 \\
\neg x_1 \vee x_2 \vee \neg x_4 \vee x_5 \\
x_1 \vee \neg x_2 \vee \neg x_4 \vee x_5 \\
\neg x_1 \vee x_2 \vee x_4 \vee \neg x_5 \\
x_1 \vee \neg x_2 \vee x_4 \vee \neg x_5 \\
x_1 \vee x_2 \vee \neg x_4 \vee \neg x_5 \\
\neg x_1 \vee \neg x_2 \vee \neg x_4 \vee \neg x_5
\end{array}$$

Fig. 2: ANF-to-CNF conversions of polynomial $x_1x_3 \oplus x_1 \oplus x_2 \oplus x_4 \oplus 1$. (Left) Karnaugh map conversion (6 CNF clauses); (Right) Tseitin-based conversion (11 CNF clauses).

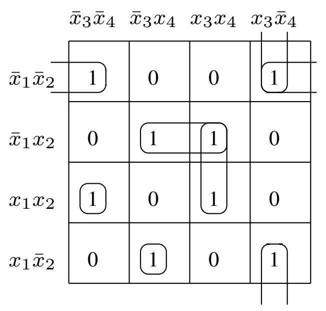

Fig. 3: Karnaugh map of polynomial $x_1x_3 \oplus x_1 \oplus x_2 \oplus x_4 \oplus 1$.

When used as a CNF preprocessor, BOSPHORUS obtains an equivalent ANF in the following manner [23]:

1) Each CNF variable is assigned a unique ANF variable;
2) Each clause is converted to a polynomial via product of negated literals.

For instance, the polynomial for the clause $\neg x_1 \vee x_2$ is $(x_1)(x_2 \oplus 1) = x_1x_2 \oplus x_1$. The resultant polynomial degree is the number of literals in each clause. More importantly, if a clause has n positive literals, there will be 2^n terms in the polynomial. To prevent such cases, we re-express the clause as a set shorter of clauses by introducing auxiliary variables *à la* converting a k-SAT to 3-SAT. We limit the number of positive literals within each of the shorter clauses to L', called the clause-cutting length. Each of the shorter clauses is then converted to polynomials as outlined above.

This CNF-to-ANF conversion is trivial, unlike that in [24]; sophisticated techniques are then applied to simplify the problem on the ANF level. In this use-case, converting problem from CNF to ANF and back to CNF give a suboptimal description of the original problem. Hence, BOSPHORUS returns the original CNF in addition to the one converted from its internal ANF representation, which includes the learnt facts.

E. Implementation

BOSPHORUS uses the following existing work:

PolyBoRi[25] To store and manipulate Boolean polynomials.

M4RI [26], [27] For efficient Gauss-Jordan elimination on Boolean matrices, necessary for XL and ElimLin.

CryptoMiniSat5 [28] This is a SAT solver equipped with conflict-driven clause learning. To extract learnt facts from this solver, we modify version 5.6.3 of the solver to exposed its APIs that extract *linear equations*.

ESPRESSO [29] For Karnaugh map simplification [30]. While ESPRESSO is a heuristic logic minimizer, it is fast and often yields close-to-optimum representations.

IV. EXPERIMENTS AND RESULTS

We run experiments on three classes of problem described in ANFs and a set of problems in CNFs. The ANF problems are round-reduced AES cipher, round-reduced SIMON cipher and weakened Bitcoin nonce finding, while the CNF problems consist of a wide variety from the SAT Competition 2017 [31]. These problems are detailed in the appendix. The experiments are conducted on a single Intel Xeon E5-2670v2 2.50GHz processor core. Each ANF or CNF is passed to BOSPHORUS, which, after learning facts using the XL–ElimLin–SAT-solver loop together with ANF propagation, will give a CNF that includes the learnt facts. A SAT solver is then used to solve the processed CNF eventually. Note that the most efficient off-the-shelf ANF solver, M4GB, has such a high memory footprint that it times out on all the instances.

We also pass the instances to the SAT solvers directly without learning facts but only converting to CNFs using BOSPHORUS if needed. We also evaluate with three different SAT solvers for the eventual solving: a minimalistic SAT solver MiniSat [16], a high-performance SAT solver Lingeling [32], and CryptoMiniSat5 [28], which natively performs Gauss-Jordan elimination.[1] We report the PAR-2 score [31] and the number of solved instances. The PAR-2 score is the sum of runtimes for solved instances and twice the timeout for unsolved instances, and a lower score is better.

For the BOSPHORUS's workflow, we use the following parameters: XL and ElimLin subsampling parameter $M = 30$, XL expansion allowance $\delta M = 4$ and degree $D = 1$, Karnaugh parameter $K = 8$, cutting lengths $L = L' = 5$, and SAT-solver conflict budget starting from $C = 10,000$, increasing up to $100,000$ in increments of $10,000$ when the learnt clauses from the SAT-solver produce no new learnt facts. Moreover, we make BOSPHORUS exit the loop and provide the solution if the SAT solver finds a satisfying assignment. We limit the total time used for each instance to 5,000 seconds, with BOSPHORUS given at most 1,000 seconds.

We only present results for selected benchmarks in Table II. The first column represents the class of benchmarks followed by the number of instances in parenthesis. For each problem class, we have two rows of results: the first without using BOSPHORUS and the second with. The third, fourth and fifth columns specify the PAR-2 score (in thousands) for MiniSat, Lingeling, and CryptoMiniSat5 respectively. The PAR-2 score

[1]The versions used are 2.2, bcj-78ebb86-180517 and 5.6.3 respectively.

TABLE II: The PAR-2 score is shown in thousands (lower is better) with, in parenthesis, the number of solved satisfiable instances plus (if any) the number of solved unsatisfiable instances. For each problem set, there are two rows of results: the first without using BOSPHORUS (labeled *w/o* in the second column) and the second with (labeled *w*). The better of the two is in bold, with preference to the number of solved instances.

Problem		MiniSat	Lingeling	CryptoMiniSat5
SR-[1,4,4,8]	w/o	4372 (89)	532 (500)	**504 (500)**
(500)	w	**1099 (489)**	**518 (500)**	507 (500)
Simon-[8,6]	w/o	**1 (50)**	**0 (50)**	**0 (50)**
(50)	w	3 (50)	3 (50)	3 (50)
Simon-[9,7]	w/o	324 (22)	**0 (50)**	**2 (50)**
(50)	w	**15 (50)**	14 (50)	14 (50)
Simon-[10,8]	w/o	500 (0)	31 (50)	45 (50)
(50)	w	**231 (34)**	**29 (50)**	**44 (50)**
Bitcoin-[10]	w/o	**4 (50)**	**9 (50)**	**8 (50)**
(50)	w	23 (50)	23 (50)	24 (50)
Bitcoin-[15]	w/o	**146 (43)**	**185 (39)**	169 (40)
(50)	w	171 (42)	220 (34)	**176 (41)**
Bitcoin-[20]	w/o	493 (1)	475 (3)	486 (2)
(50)	w	**482 (2)**	**471 (4)**	**477 (3)**
SAT-2017	w/o	2105 (75+38)	2006 (70+56)	1764 (89+63)
(310)	w	**2153 (72+42)**	**2070 (70+57)**	**1674 (98+77)**
SAT-2017	w/o	2045 (15+ 7)	1738 (29+26)	1689 (30+32)
(219)	w	**1981 (18+11)**	**1756 (29+27)**	**1543 (40+46)**

for the case of using BOSPHORUS includes time taken by BOSPHORUS. The Simon, Bitcoin and SAT-2017 benchmark classes are listed in increasing difficulty.

For the instances from SR-[1,4,4,8], BOSPHORUS allows a significantly more solved instances for MiniSat, and it provides similar PAR-2 scores for Lingeling and CryptoMiniSat5 even while including its overhead. Similar observations can be made for the harder Simon instances, though the overhead of BOSPHORUS is now clearly visible in Simon-[9,7]. With Bitcoin, BOSPHORUS does not always help, but the effect of its overhead to the PAR-2 scores diminishes with the harder instances. One way to study when BOSPHORUS helps is to run it with different parameters. For the SAT-2017 CNF instances, BOSPHORUS does provide useful information to the solvers, especially for the UNSAT instances.

V. DISCUSSION

While BOSPHORUS can be used as a CNF preprocessor, it is in fact a flexible reasoning framework on Boolean or GF(2) variables in the following sense. First, for satisfiable problems, the SAT solver collapse onto *one* solution, while BOSPHORUS can continuously constrain the solution space without committing to one particular solution. Second, for unsatisfiable problems, the conclusion can be reached by either any of the ANF techniques giving $1 = 0$ or by the SAT solver giving UNSAT. Third, any of the solving techniques in the workflow can be improved with minimal impact on the other techniques because the retained facts do not increase the complexity of the equations. Fourth, it is relatively easy to include new solving techniques by plugging them as components into the workflow, for example, lookahead SAT solvers [33] and Buchberger's algorithm [3]. In fact, using the Buchberger's algorithm as a preprocessor for SAT solving has previously been proposed [24], but, with BOSPHORUS, it may now be applied in an iterative manner together with other solving techniques.

To conclude, we have proposed and implemented a tool named BOSPHORUS that iteratively applies eXtended linearization, ElimLin and conflict-bounded SAT solving together with ANF propagation in order to learn additional facts to augment the original problem. The experiments on selected ANF and SAT problems have shown that this approach can help solve more problems in a shorter time, particularly for the harder instances.

Acknowledgments: We thank Joshua Wong for running the M4GB experiments, Sze Ling Yeo for her Simon encoding, and Volodymyr Skladanivskyy for his SHA256 encoding. We also thank Karsten Nohl at Security Research Labs for supporting some of this work. This work was supported in part by NUS ODPRT Grant R-252-000-685-133 and AI Singapore Grant R-252-000-A16-490.

REFERENCES

[1] NIST, "Advanced Encryption Standard (AES)," *FIPS PUB 197*, 2001.
[2] ——, "Secure Hash Standard (SHS)," *FIPS PUB 180–4*, 2015.
[3] B. Buchberger, "Bruno Buchbergers PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal," *Journal of Symbolic Computation*, 2006.
[4] J.-C. Faugere, "A new efficient algorithm for computing Gröbner bases (F4)," *Journal of pure and applied algebra*, 1999.
[5] J. C. Faugère, "A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)," in *Proceedings of ISSAC*, 2002.
[6] R. H. Makarim and M. Stevens, "M4GB: an efficient Gröbner-basis algorithm," in *Proceedings of ISSAC*, 2017.
[7] J.-C. Faugère, "FGb: a library for computing Gröbner bases," in *Proceedings of ICMS*, 2010.
[8] W. Bosma, J. Cannon, and C. Playoust, "The Magma algebra system i: The user language," *Journal of Symbolic Computation*, 1997.
[9] N. Courtois, A. Klimov, J. Patarin, and A. Shamir, "Efficient algorithms for solving overdefined systems of multivariate polynomial equations," in *Proceedings of Eurocrypt*, 2000.
[10] N. T. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations," in *Proceedings of Asiacrypt*, 2002.
[11] N. T. Courtois and G. V. Bard, "Algebraic cryptanalysis of the Data Encryption Standard," in *IMA International Conference on Cryptography and Coding*, 2007.
[12] N. T. Courtois, P. Sepehrdad, P. Sušil, and S. Vaudenay, "ElimLin algorithm revisited," in *Proceedings of FSE*, 2012.
[13] S. Gao, Y. Guan, and F. Volny IV, "A new incremental algorithm for computing Gröbner bases," in *Proceedings of ISSAC*, 2010.
[14] M. R. Prasad, A. Biere, and A. Gupta, "A survey of recent advances in SAT-based formal verification," *Proceedings of STTT*, 2005.
[15] J. a. P. M. Silva and K. A. Sakallah, "GRASP - a new search algorithm for satisfiability," in *Proceedings of CAV*, ser. ICCAD '96. IEEE Computer Society, 1996, pp. 220–227.
[16] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proceedings of SAT*, 2003.
[17] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *Proceedings of SAT*, F. Bacchus and T. Walsh, Eds. Springer Berlin Heidelberg, 2005, pp. 61–75.
[18] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings DAC*, 2001.
[19] K. S. Meel, M. Y. Vardi, S. Chakraborty, D. J. Fremont, S. A. Seshia, D. Fried, A. Ivrii, and S. Malik, "Constrained sampling and counting: Universal hashing meets SAT solving," in *AAAI Workshop: Beyond NP*, 2016.
[20] K. S. Meel, *Constrained Counting and Sampling: Bridging the gap between Theory and Practice*. Rice University, 2017, Ph.D. Thesis.

[21] H. Zhang, "SATO: An efficient prepositional prover," in *Proceedings of CADE*, 1997.
[22] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of reasoning*, 1983.
[23] J. Hsiang, "Refutational theorem proving using term-rewriting systems," *Artificial Intelligence*, 1985.
[24] C. Condrat and P. Kalla, "A Gröbner basis approach to CNF-formulae preprocessing," in *Proc. of TACAS*, 2007.
[25] M. Brickenstein and A. Dreyer, "PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials," *Journal of Symbolic Computation*, 2009.
[26] M. Albrecht and G. Bard, *The M4RI Library – Version 20121224*, The M4RI Team, 2012. [Online]. Available: http://m4ri.sagemath.org
[27] M. Albrecht, G. Bard, and C. Pernet, "Efficient dense Gaussian elimination over the finite field with two elements," 2011, arXiv:1111.6549v1[cs.MS].
[28] M. Soos, "The CryptoMiniSat 5 set of solvers at SAT competition 2016," in *SAT Competition 2016 – Solver and Benchmark Descriptions*, 2016.
[29] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic minimization algorithms for VLSI synthesis*, 1984.
[30] M. Karnaugh, "The map method for synthesis of combinational logic circuits," *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 1953.
[31] T. Balyo, M. Heule, and M. Järvisalo, Eds., *SAT Competition 2017 – Solver and Benchmark Descriptions*. University of Helsinki, 2017, vol. B-2017-1.
[32] A. Biere, "CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017," in *SAT Competition 2017 – Solver and Benchmark Descriptions*, 2017.
[33] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*, 2009.
[34] C. Cid, S. Murphy, and M. J. Robshaw, "Small scale variants of the AES," in *Proceedings of FSE*, 2005.
[35] The Sage Developers, *SageMath, the Sage Mathematics Software System (Version 8.1)*, 2017, http://www.sagemath.org.
[36] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, "The Simon and Speck lightweight block ciphers," in *Proceedings of DAC*, 2015.
[37] N. Courtois, T. Mourouzis, G. Song, P. Sepehrdad, and P. Susil, "Combined algebraic and truncated differential cryptanalysis on reduced-round Simon," in *Proceedings of SECRYPT*, 2014.
[38] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Tech. Rep., 2008.

APPENDIX

A. Round-reduced AES cipher — 500 instances

We obtain a parameterized ANF encoding of AES [34] from SageMath [35]. Using parameters $(n, r, c, e) = (1, 4, 4, 8)$, we generate 500 ANF instances for 1-round AES. First, 500 random pairs of plaintext (P) and key (K) bits are generated and simulated to yield the corresponding ciphertext (C) bits. The resultant ANF has 800 variables and 1120 equations — 864 equations and 256 bit assignments from (P, C).

B. Round-reduced Simon cipher — 50 instances per (n, r)

Simon [36] is a family of lightweight Feistel-based block ciphers. The round functions are described in conjunction and exclusive-OR of bits, allowing a straightforward ANF encoding; see Fig. 4. This set of benchmarks are reduced rounds Simon32/64 with multiple plaintext-ciphertext pairs encoded under the same randomly generated secret key.

Simon32/64 takes a 32-bit plaintext (P) and a 64-bit key to return a 32-bit ciphertext. For each instance, we generate $n \leq 17$ plaintexts with low hamming distance as per the Similar Plaintexts/Random Ciphertexts (SP/RC) setting in [37]. Concretely, the first plaintext P_1 is uniformly sampled while

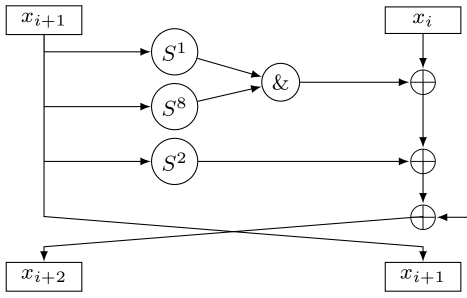

Fig. 4: One Fiestel round of Simon cipher. Diagram from [36].

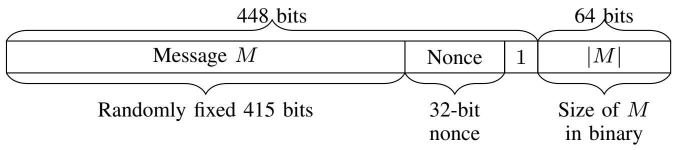

Fig. 5: Our nonce-finding setup.

we toggle the ith in the right-half of P_1, for $i \in \{2, \ldots, n\}$. This set of problems is parameterized by (n, r), where n is the number of plaintexts, and r is the number of rounds.

C. Cryptographic hash functions — 50 instances per k

Recently, Cryptographically secure hash functions have been used to serve as proof-of-work in blockchains and cryptocurrencies, of which Bitcoin is an example. Bitcoin [38] uses SHA256, a hash function in the SHA-2 hash family [2].

We consider a *weakened* version of the Bitcoin block hashing algorithm. Let M be a 512-bit input message, and H be a 256-bit hash output. We randomly set the first 415 bits of M, allow the next 32-bit nonce to be free (but to be determined), and pad according to SHA padding (add '1', then encode $|M| = 448$ in the next 64 bits). Given k, the challenge is then to solve for a suitable 32-bit nonce of M that results in a hash H with the first k bits being 0. We construct challenges in this manner because Bitcoin uses 32-bit nonces to solve for hashes starting with varying k zeroes. See Fig. 5 for an illustration. We generate instances for $k = \{10, 15, 20\}$ using the generic ANF encoding available at https://github.com/vsklad/cgen.

D. Instances from SAT 2017 Competition

We preprocess `g2-hwmcc15deep-beemfwt4b1-k48` and `g2-hwmcc15deep-beemlifts3b1-k29` using CryptoMiniSat5 to reduce the number of variables to less than 1,048,574 variables, which is the maximum number of variables that the POLYBORI data structure can handle on our platforms. We omit the 40 CNFs with names of the pattern `g2-T*` because they each have too many variables even after the preprocessing. We also omit `mp1-bsat222-777` because it is not a well-formed DIMACS file. Hence, we experiment on 310 instances altogether. From these, we select difficult instances: using the runtime of MiniSat (without BOSPHORUS) as a proxy difficulty measure, we select the 219 that requires more than 2,500 seconds.