Entailing Generalization Boosts Enumeration

Dror Fried
Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel

Alexander Nadel
Intel Corporation, Israel and Faculty of Data and Decision Sciences, Technion, Haifa, Israel

Roberto Sebastiani
DISI, University of Trento, Italy

Yogev Shalmon
Intel Corporation, Israel and The Open University of Israel, Ra'anana, Israel

Abstract

Given a combinational circuit Γ with a single output o, AllSAT-CT is the problem of enumerating all solutions of Γ. Recently, we introduced several state-of-the-art AllSAT-CT algorithms based on *satisfying generalization*, which generalizes a given total Boolean solution to a smaller ternary solution that still satisfies the circuit. We implemented them in our open-source tool `HALL`. In this work we draw upon recent theoretical works suggesting that utilizing generalization algorithms, which can produce solutions that *entail* the circuit without *satisfying* it, may enhance enumeration. After considering the theory and adapting it to our needs, we enrich `HALL`'s AllSAT-CT algorithms by incorporating several newly implemented generalization schemes and additional SAT solvers. By conducting extensive experiments we show that *entailing* generalization substantially boosts `HALL`'s performance and quality (where *quality* corresponds to the number of reported generalized solutions per instance), with the best results achieved by combining *satisfying* and entailing generalization.

2012 ACM Subject Classification Mathematics of computing → Solvers

Keywords and phrases Generalization, Minimization, Prime Implicant, AllSAT, SAT, Circuits

Digital Object Identifier 10.4230/LIPIcs.SAT.2024.13

Supplementary Material *Software*: `https://github.com/yogevshalmon/allsat-circuits`
archived at `swh:1:dir:9c4d2c70cd996ca68c8837178682cb4c32cd4e20`

Funding We acknowledge the support of the MUR PNRR project FAIR – Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU. The work was partially supported by the project "AI@TN" funded by the Autonomous Province of Trento. Also funded in part by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HaDEA). Neither the European Union nor the granting authority can be held responsible for them. Grant Agreement no. 101120763 - TANGO.
Yogev Shalmon: This research was supported by The Open University of Israel's Research Fund.

Acknowledgements We are grateful to Ben Emanuel for helpful discussions which played an important role in shaping our research. Also, we thank the anonymous reviewers for their comments and useful suggestions.

1 Introduction

Enumerating the solutions of a given propositional formula is often a required task in computer science [20, 11, 22, 49, 52]. In *AllSAT-CT*, the formula is provided in a form of a combinational circuit $\Gamma = \langle I, G, o\rangle$ with inputs I, gates G and a single output o. Then, the goal is to enumerate all the possible assignments to Γ's inputs, for which Γ's output is 1 (see Fig. 1 for an example). AllSAT-CT's applications include model checking [28, 17, 18] and

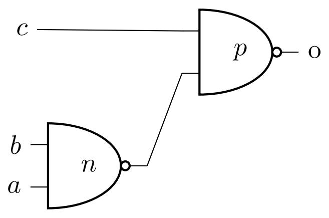
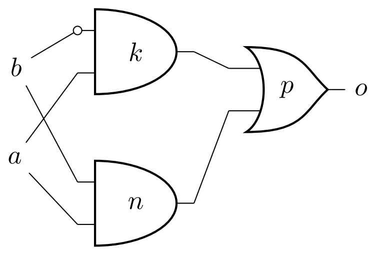


Figure 1 The circuit $\Gamma = \langle I = \{a, b, c\}, G = \{n \leftrightarrow a \wedge b, p \leftrightarrow \neg n \wedge c\}, o \equiv \neg p\rangle$ is shown. An AllSAT-CT solver could return the following two solutions: $\sigma_1 \equiv \{c := 0\}$ and $\sigma_2 \equiv \{a := 1; b := 1\}$.



(a) $\Gamma = \langle I = \{a, b\}, G = \{n \leftrightarrow a \wedge b, k \leftrightarrow a \wedge \neg b, p \leftrightarrow n \vee k\}, o \equiv p\rangle$.

$$C_1 = (n \vee \neg a \vee \neg b),$$
$$C_2 = (\neg n \vee a), C_3 = (\neg n \vee b),$$
$$C_4 = (k \vee \neg a \vee b),$$
$$C_5 = (\neg k \vee a), C_6 = (\neg k \vee \neg b),$$
$$C_7 = (\neg p \vee k \vee n),$$
$$C_8 = (p \vee \neg k), C_9 = (p \vee \neg n)$$

(b) Encoding G to CNF.

Figure 2 An example where UC generalization returns an e-hard solution (that is, an e-generalization which cannot be subsumed by any s-generalization). Let $\sigma \equiv \{a := 1, b := 1\}$ be the solution to Γ, depicted in Fig. 2a, we are interested to generalize to $\tau \equiv \{a := 1\}$. τ is an e-generalization of σ, since with $a = 1$ the output must be 1 whether b is assigned 1 or 0, but not an s-generalization, since ternary simulating τ would assign X to k, n and o. Clearly, τ is also not subsumed by any other solution, so τ is e-hard. The translation of $\neg\Gamma$ to CNF using Tseitin encoding would contain the clauses in Fig. 2b and the unit clause $(\neg p)$ representing the negation of the output. Propagating $\neg p$ by the SAT solver would imply $\neg n$ and $\neg k$ in the clauses C_8 and C_9. One can now see that assuming $a = 1$ is sufficient to get a conflict between C_1 and C_4, hence the unit cube $Q = a$ could potentially be returned by the solver as the UC, which induces $\tau^Q \equiv \{a := 1\}$ as required.

ATPG [9, 48]. Moreover, we apply AllSAT-CT solving in our industrial practice for Static Timing Analysis (STA) [46, 14], which is a crucial step in circuit design that validates the timing of a circuit by checking all possible paths for timing violations.

In a recent work of ours [14], we have introduced several anytime AllSAT-CT algorithms that work by iteratively retrieving a solution, generalizing it, reporting it to the user, and subsequently blocking it. These algorithms, implemented into an open-source tool called `HALL`, exhibited state-of-the-art performance and quality (where *quality* corresponds to the number of reported generalized solutions). Increasing the quality is vital in AllSAT-CT, particularly in STA, where testing as few potential timing violations as possible is required. In this work, we have substantially improved both performance and quality of `HALL` on a wide range of benchmarks, mainly by upgrading `HALL`'s generalization component, leveraging the insights outlined below.

We first discuss generalization. Given a circuit Γ and its total Boolean solution $\sigma(I)$, it is often required to *generalize* σ to a small *ternary* solution by replacing as many Boolean values as possible by X's (don't cares), while making sure that the generalized σ is still a solution to Γ. Generalization is a variant of (prime) implicant generation, the latter extensively studied since the 1950th [39, 27, 44, 19, 7, 10, 38, 23], where in generalization there is a starting solution that must be subsumed by the resulting implicant. Since the early 2010s, generalization has been widely used as a core component in IC3 (aka PDR) model checking algorithm and its derivatives [6, 8, 12, 51, 21]. A careful look, however, reveals that the definition of generalization is ambiguous. Indeed, since generalization generates ternary assignments, to define it one must answer the following question: what does it mean for a given ternary input assignment $\tau(I) : I \mapsto \{0, 1, X\}$ to serve as a solution to the circuit? One possibility would be as follows. Every $\tau(I)$ can be expanded to the assignment $\tau^S(I \cup G \cup \{o\})$ by propagating $\tau(I)$ to every gate and the output by *ternary simulation* (see Sect. 2). We then say that τ *satisfies* Γ (denoted by $\tau \approx \Gamma$), if $\tau^S(o) = 1$. For example, in Fig. 1, we have $\{c := 0\} \approx \Gamma$ and $\{a := 1; b := 1\} \approx \Gamma$ (assuming any omitted variables in ternary assignments are assigned X). One could have defined that a ternary τ is a solution to Γ iff $\tau \approx \Gamma$. Another option, however, inequivalent to satisfaction, is to define a ternary τ to comprise a solution to Γ iff τ *entails* the circuit, where τ *entails* Γ (denoted by $\tau \models \Gamma$), if $\rho^S(o) = 1$ for *any* ρ which substitutes *every* X in τ by *any* Boolean value. To understand why entailment is preferable to satisfaction for solution definition, consider the circuit Γ in Fig. 2a (for now ignore Fig. 2's caption, discussed in Sect. 3) and the assignment $\psi \equiv \{a := 1\}$. ψ qualifies as a solution to Γ through both intuitive understanding and our entailment-based solution definition, since ternary simulation renders either $k = 1$ or $n = 1$ for either $b = 0$ or $b = 1$, respectively, so $o = 1$ is implied no matter what. However, ψ does *not* satisfy Γ, since, given $b = X$, ternary simulation would assign X to both the gates k and n and then the output too.

The core of our analysis is based on our previously unpublished work [42] and later follow-ups [30, 31], which made the key distinction between entailment and satisfaction and surmised that integrating duality [15, 29]-based generalization algorithms, expected to output solutions which entail the formula without satisfying it, should boost enumeration. Our work, however, is the first to exhibit how to capitalize on this observation to advance the state of the art in enumeration empirically, thereby bridging the gap between theory and practice (the duality-based model counter `dualiza` [29] can also solve AllSAT-CT, but Sect. 5 shows that it is inefficient).

As such, we present in Sect. 3 three distinct generalization definitions, ordered in a hierarchy, including the most powerful *entailing (e-)generalization* where the generalized τ has to merely entail Γ, followed by *satisfying (s-)generalization* where τ must satisfy Γ, itself followed by an even more restricted *gate (g-)generalization* (intended to argue that generalizing after reducing the circuit to clauses is inefficient). The first two definitions are based on our previously unpublished work [42], while the third one is novel. We then classify commonly used generalization algorithms based on our hierarchy and observe that duality-based *Unsatisfiable Core-based (UC) generalization* [8] can potentially actualize the advantage of e-generalization.

Next, we leverage our analysis to boost `HALL`, so far based on (forward) *ternary generalization* [41, 12], restricted to s-generalization. Substituting ternary by UC generalization substantially improves `HALL`'s performance and quality, further improved by combining ternary and UC generalization (as UC generalization does not guarantee the smallest cardinality). Additionally, we study and compare the impact of the following newly implemented components in `HALL`: the SAT solvers `CaDiCaL` [4], `MergeSat` [25] and `CryptoMiniSAT` [45] (added alongside `IntelSAT` [35]), *backward* ternary generalization [41] (aka *justification* [43]) and UC generalization [8] with or without minimization [40, 32]. In what follows, Sect. 4 discusses AllSAT-CT. Sect. 5 is dedicated to experimental evaluation. In Sect. 6 we conclude.

2 Preliminaries

We briefly review the relevant syntax of Boolean logic. Let V be the set of Boolean variables. A *literal* l is either a variable $v \in V$ or its negation $\neg v$. A *clause/cube* is a disjunction/conjunction of literals. A formula $F(V)$ is in *Conjunctive/Disjunctive Normal Form (CNF/DNF)* if it is a conjunction/disjunction of clauses/cubes. A (combinational Boolean single-output) *circuit* $\Gamma = \langle I = \{v_1, \cdots v_n\}, G = \{v_{n+1}, \cdots v_{n+m}\}, o \in \{v_{n+m}, \neg v_{n+m}\}\rangle$ is a tuple, where I are the *inputs*, G are the *gates* and o is the *output*. Every gate comprises the formula $v_k \leftrightarrow (l_i \wedge l_j)$, where $i, j < k$ and l_i, l_j are literals of variables v_i and v_j respectively (using only $\wedge$ operator does not restrict the generality [5]). Tseitin encoding [50] converts a given circuit Γ to a CNF formula by translating every gate $v \leftrightarrow l_1 \wedge l_2$ to three clauses $(v \vee \neg l_1 \vee \neg l_2) \wedge (\neg v \vee l_1) \wedge (\neg v \vee l_2)$ and adding the unit clause (o) to assert the output.

For brevity, we skip the standard Boolean logic semantics. *Ternary logic* [37] extends Boolean logic with an additional value called *don't-care (X)*. Formally, a *ternary* assignment $\tau : V \mapsto \{0, 1, X\}$ assigns each variable to one of the *ternary values* $\{0, 1, X\}$. The *cardinality* $|\tau|$ of a ternary assignment τ is the number of variables in τ assigned 0 or 1 (inducing an order relation between assignments). A ternary assignment is also a *total Boolean assignment* iff it has the maximal cardinality. To evaluate a formula in Boolean logic syntax under a ternary assignment τ, one can use Boolean logic semantics extended by the rules $(\neg X \equiv X)$, $(X \wedge 1 \equiv X)$, $(X \wedge 0 \equiv 0)$ and $(X \wedge X \equiv X)$.

Ternary simulation propagates a given ternary assignment to the inputs τ across the gates all the way to the output:

▶ **Definition 1** (Ternary Simulation [41, 16]). *Given a circuit $\Gamma = \langle I, G, o\rangle$ and a ternary assignment $\tau(I) : I \mapsto \{0, 1, X\}$ to Γ's inputs,* ternary simulation *transforms τ to the assignment $\tau^S(\{v_1 \dots v_{n+m}\})$, where $\tau^S(v) := \tau(v)$ for every input $v \in I$, and for every gate $v_k \leftrightarrow (l_i \wedge l_j)$, we have $\tau^S(v_k) := \tau^S(l_i) \wedge \tau^S(l_j)$.*

For brevity, we omit variables assigned X when specifying ternary assignments. We say that a ternary assignment $\rho(I)$ *subsumes* the ternary assignment $\tau(I)$, denoted by $\rho \subseteq \tau$, if $\tau(v) = \rho(v)$ for every $v : \rho(v) \in \{0, 1\}$. We say that $\rho(I)$ *strictly subsumes* $\tau(I)$, denoted by $\rho \subset \tau$, if $\rho \subseteq \tau$ and $|\rho| < |\tau|$. For example, $\{x_1 := 1\} \subset \{x_1 := 1, x_2 := 0\}$. A ternary assignment $\tau(I)$ naturally induces the cube D^τ containing v wherever $\tau(v) = 1$ and $\neg v$ wherever $\tau(v) = 0$ (variables assigned X's are skipped). Similarly, a cube $D(I)$ induces a ternary assignment, denoted by τ^D, in which $\tau(v) = 1$ for $v \in D$, $\tau(v) = 0$ for $\neg v \in D$ and $\tau(v) = X$ if $v, \neg v \notin D$. For example, given $I = \{a, b, c\}$, $\tau(I) \equiv \{a := 1, b := 0\}$ induces the cube $D^\tau = a \wedge \neg b$, while the cube $D(I) = a \wedge \neg b$ induces $\tau^D \equiv \{a := 1, b := 0\}$.

Given a CNF formula F, a SAT solver decides whether F is satisfiable. Many SAT solvers are *incremental* [13, 36]: they can be invoked multiple times, where, for every new query SAT(F, A), the SAT solver also receives a cube of *assumption literals (assumptions)* A, which hold only for the current query. The solver then decides whether $F \wedge A$ is satisfiable (where F contains all the clauses provided so far). If $F \wedge A$ is unsatisfiable, SAT(F, A) returns an *Unsatisfiable Core (UC)*, that is, a cube $A' \subseteq A$, such that $F \wedge A'$ is still unsatisfiable [13].

3 The Generalization Hierarchy

Recall from Sect. 1 the following definitions of a ternary assignment $\tau(I) : I \mapsto \{0, 1, X\}$ *satisfying* ($\approx$) and *entailing* ($\models$) a given circuit $\Gamma = \langle I, G, o\rangle$:

1. τ *satisfies* Γ (denoted by $\tau \approx \Gamma$), if $\tau^S(o) = 1$,
2. τ *entails* Γ (denoted by $\tau \models \Gamma$), if $\rho^S(o) = 1$ for *any* ρ which substitutes *every* X in τ by *any* Boolean value.

We define a solution to the least restrictive option sufficient for real-world applications (e.g., AllSAT-CT or PDR): $\tau(I)$ is a *solution* to Γ iff $\tau \models \Gamma$.

In addition, we say that τ *satisfies the gate* $v \in G$, if $\tau^S(v) \neq X$, and that τ *gate-satisfies* Γ if τ satisfies Γ and every gate in Γ. Def. 2 offers three alternatives for defining generalization, where *any* generalization τ must subsume the given total Boolean solution σ.

▶ **Definition 2** (G-,s-,e-generalization). *Given a circuit* $\Gamma = \langle I, G, o \rangle$ *and its total Boolean solution* $\sigma(I) \approx\!\!\models \Gamma$*, a ternary solution* $\tau(I) \models \Gamma : \tau(I) \subseteq \sigma(I)$ *is a:*

- gate (g-) generalization *of* σ *if* τ *gate-satisfies* Γ *(that is,* $\tau \approx\!\!\models \Gamma$ *and* $\forall v \in G : \tau^S(v) \neq X$*)*
- satisfying (s-) generalization *of* σ *if* $\tau \approx\!\!\models \Gamma$
- entailing (e-) generalization *of* σ *if* $\tau \models \Gamma$

E-generalization is the least restrictive one, merely requiring τ to be Γ's solution. S-generalization requires τ to satisfy the circuit, while g-generalization additionally has τ satisfying every single gate. We denote the sets of all the g-, s- and e- generalizations for a given circuit Γ and a total Boolean solution $\sigma \approx\!\!\models \Gamma$ by $G(\Gamma, \sigma)$, $S(\Gamma, \sigma)$ and $E(\Gamma, \sigma)$, respectively. Towards separating between g- and s-generalization as well as between s- and e-generalization, Def. 3 introduces the notions of s-hard and e-hard solutions.

▶ **Definition 3** (S-hard, e-hard). *Given a circuit* $\Gamma = \langle I, G, o \rangle$ *and its total Boolean solution* $\sigma(I) \approx\!\!\models \Gamma$*, a ternary solution* $\tau(I) \models \Gamma$ *is*

- s-hard *if* $\tau \in S(\Gamma, \sigma)$*, but for every* $\rho \subsetneq \tau : \rho \notin G(\Gamma, \sigma)$
- e-hard *if* $\tau \in E(\Gamma, \sigma)$*, but for every* $\rho \subseteq \tau : \rho \notin S(\Gamma, \sigma)$

Lemma 4 below presents the generalization hierarchy. It shows that e-generalization is *more powerful (denoted by* $\gg$*)* than s-generalization in the following sense: every s-generalization is an e-generalization, but there exists an e-hard solution τ which separates between e- and s-generalization (that is, τ is an e-generalization, but no $\rho \subseteq \tau$ is an s-generalization). Similarly, s-generalization $\gg$ g-generalization. The generalization hierarchy is illustrated in Fig. 3.

▶ **Lemma 4** (e-generalization $\gg$ s-generalization $\gg$ g-generalization). *The lemma is threefold:*

I. *For every* $\Gamma = \langle I, G, o \rangle$ *and total Boolean* $\sigma(I) \approx\!\!\models \Gamma$*:* $G(\Gamma, \sigma) \subseteq S(\Gamma, \sigma) \subseteq E(\Gamma, \sigma)$.
II. *There exists an s-hard solution for some* $\Gamma = \langle I, G, o \rangle$ *and a total Boolean* $\sigma(I) \approx\!\!\models \Gamma$.
III. *There exists an e-hard solution for some* $\Gamma = \langle I, G, o \rangle$ *and a total Boolean* $\sigma(I) \approx\!\!\models \Gamma$.

Proof. I is straightforward. For II, consider Fig. 1, where $\{c := 0\}$ is s-hard, given $\sigma = \{a := 1; b := 1; c := 0\}$ (since the output is satisfied by ternary simulating τ, but gate n is not). For III, consider Fig. 2a, where $\{a := 1\}$ is e-hard, given $\sigma = \{a := 1, b := 1\}$. ◀

Next, we classify popular generalization algorithms, based on the hierarchy in Lemma 4. Consider any *Tseitin generalization* algorithm which translates the circuit to a CNF using Tseitin encoding and generalizes at CNF level by any algorithm (see, e.g. [7, 10, 49]) that turns as many variables as possible to don't cares, while still guaranteeing that every clause is satisfied. Such algorithms can only generate g-generalizations. Indeed, in Tseitin encoding, every gate $v \leftrightarrow l_1 \wedge l_2$ is translated to $(v \vee \neg l_1 \vee \neg l_2) \wedge (\neg v \vee l_1) \wedge (\neg v \vee l_2)$. Hence, the variable v representing the gate must be assigned a Boolean value, since, otherwise, one or two of the three clauses (depending on the values of l_1 and l_2) would have been left unsatisfied.

Let (forward) *ternary generalization* [41, 12] be the algorithm that generalizes a given solution by iteratively assigning every input v to X iff propagating $v := X$ by ternary simulation still sets the output to 1. While, in principle, the inputs can be visited in any order, our implementation visits the inputs in their order from 1 to n.

Note that ternary generalization can also be carried out *backwards* [41], where *backward ternary generalization* is also known as *justification* [43]. Briefly speaking, backward ternary generalization traverses the circuit's gates in a reversed order (starting from the output).

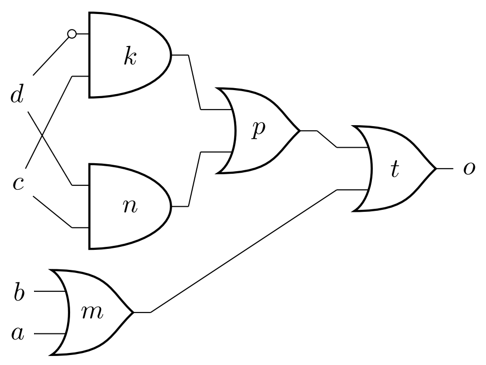


Figure 3 Illustrating the generalization hierarchy-related concepts on the circuit $\Gamma = \langle I = \{a,b,c,d\}, G = \{m \leftrightarrow a \vee b, n \leftrightarrow c \wedge d, k \leftrightarrow c \wedge \neg d, p \leftrightarrow n \vee k, t \leftrightarrow p \vee m\}, o \equiv t\rangle$. All of the following assignments are solutions to Γ: $\sigma \equiv \{a := 1, b := 1\, c := 1, d := 1\}$, $\rho \equiv \{b := 1, c := 1, d := 1\}$, $\tau \equiv \{b := 1, c := 1\}$ and $\mu \equiv \{c := 1\}$, where σ is the only Boolean solution, and we have $\mu \subset \tau \subset \rho \subset \sigma$ by construction. Observe that μ is an e-hard e-generalization of σ, τ is an s-hard s-generalization of σ, whereas ρ is a g-generalization of σ (but ρ is not an s- nor an e-generalization).

Whenever a gate whose output is not X is encountered, the algorithm tries to convert one of its inputs to X, whenever possible (e.g., for an $\wedge$-gate, whose output and inputs are all 0, one of the inputs can be converted to X).

Ternary generalization (both forward and backward) can generate s-hard solutions (e.g., it could generalize $\{a := 1; b := 1; c := 0\}$ to $\{c := 0\}$ in Fig. 1), but not e-hard solutions, since it uses ternary simulation for establishing satisfiability. Same holds for *dual-rail generalization* [14, 43], which applies generalization at CNF level but using ternary-logic-simulating dual-rail encoding. Specifically, in dual-rail encoding, every variable v in the original circuit is mapped to two Boolean *dual-rail* variables (v^+, v^-) in the resulting CNF, where assigning both v^+ and v^- to 0 corresponds to assigning the original v to a don't-care. Then, one can guide the SAT solver to return a generalized solution by applying anytime MaxSAT-inspired heuristics [33, 34] to increase the number of don't-cares assigned to the circuit inputs (that is, the number of 0's assigned to their respective dual-rail variables). In line with our analysis, state-of-the-art AllSAT-CT algorithms are substantially faster with ternary or dual-rail generalization than with Tseitin generalization [14].

Finally, recall UC generalization [8] (its predecessors being implication graph-based approaches [47, 28, 40]). Given a circuit $\Gamma = \langle I, G, o\rangle$, let the *dual circuit* $\neg\Gamma$ be $\langle I, G, \neg o\rangle$. Let $\sigma(I) : I \mapsto \{0,1\}$ be Γ's total Boolean solution. Note that σ does *not* satisfy $\neg\Gamma$. Let $\neg F$ be a conversion of $\neg\Gamma$ to CNF using Tseitin encoding. *Unsatisfiable Core-based (UC) generalization* [8] generalizes $\sigma(I)$ to τ^Q, where Q is the unsatisfiable core (cube), returned by the query $\mathrm{SAT}(\neg F, D^\sigma)$. For example, $\sigma \equiv \{a := 1, b := 1\}$ is a solution to Γ in Fig. 2a, hence $\mathrm{SAT}(\neg F, a \wedge b)$ must return UNSAT, and the example UC a would translate to $\tau \equiv \{a := 1\}$, which generalizes σ. UC generalization guarantees e-generalization as substituting X's in τ^Q by any Boolean values and ternary simulating must render $o = 1$, otherwise $Q \wedge \neg F$ would have been satisfiable. Crucially, unlike the other algorithms, UC generalization can generate e-hard solutions: see Fig. 2 for a detailed example. One can also minimize the UC [40, 32].

4 Generalization-based Enumeration Algorithms

Given a circuit $\Gamma = \langle I, G, o\rangle$, an AllSAT-CT solver returns a DNF formula $Q(I)$, where for every *solution cube* $D(I) \in Q(I)$, we have $\tau^D \models \Gamma$, while $G \wedge o$ and $Q(I)$ are logically equivalent. We next review the AllSAT-CT algorithms from [14] and introduce our new UC generalization-based algorithms `CORE`, `ROC` and `CARMA`. All the algorithms are implemented within the well-known *blocking* framework, which repeatedly enumerates, generalizes and blocks the solutions [28] (a correctness proof can be found in [28]). This work focuses on non-disjoint solving (i.e., a total Boolean solution can be subsumed by multiple solutions), since disjoint solving [52], although supported by `HALL`, would be impractical for AllSAT-CT applications in model checking [28, 17, 18], ATPG [9, 48] and STA [46, 14]. As a side note, AllSAT-CT is simpler than finding *all* the prime implicants [44, 38, 23], since we only need a *subset* of the (not-necessarily-prime) implicants which subsume every total Boolean solution.

Consider Alg. 1 that presents `TALE` from [14] and our novel `CORE` and `ROC` algorithms. To recall `TALE` let us follow Alg. 1 in `TALE` mode ($\mathtt{A} = \mathtt{TALE}$). First, the algorithm converts the given circuit to CNF by applying the Tseitin encoding and provides the CNF as an input to an incremental SAT solver instance `plain` (line 1). Line 3 initializes the DNF Q that will contain all the solutions. Then, the algorithm starts to iteratively produce cubes in the following way. It queries `plain` to get a total Boolean solution $\sigma \models\!\approx \Gamma$ (line 5), applies ternary generalization (the forward version by default) to generalize σ (line 6), updates DNF Q with the cube U induced by σ (lines 8 and 12) and blocks U in `plain` (line 13).

Algorithm 1 Three AllSAT-CT algorithms: `TALE`, `CORE` and `ROC`

```
    Input: Circuit $\Gamma = \langle I, G, o \rangle$        Input: $\mathtt{A} \in \{\mathtt{TALE}, \mathtt{CORE}, \mathtt{ROC}\}$
    Output: DNF $Q(I)$
 1: plain := CNFTseitin($\Gamma$)                                        ▷ Initialize plain SAT instance
 2: if A $\neq$ TALE then dual := CNFTseitin($\neg\Gamma$)                ▷ Initialize dual SAT instance, if required
 3: $Q$ := {}                    ▷ Initializing the DNF $Q$, which will contain all the solutions, to be empty
 4: while not UNSAT(plain) do
 5:     $\sigma$ := SAT(plain)
 6:     if A $\neq$ CORE then $\sigma$ := TernaryGeneralize($\sigma, \Gamma$)        ▷ Ternary generalization
 7:     if A = TALE then
 8:         $U := D^\sigma$
 9:     else
10:         $U$ := SAT(dual, $D^\sigma$)                                          ▷ Fetch the UC
11:         forall $a \in U$: if SAT(dual, $U \setminus \{a\}$) is UNSAT then $U := U \setminus \{a\}$   ▷ Minimize the UC
12:     $Q := Q \vee U$                                            ▷ $Q$ is updated by the cube $U$
13:     plain := plain $\wedge \neg U$                                    ▷ Blocking $U$ in plain
14: return $Q$                                                ▷ $Q$ is not guaranteed to be disjoint
```

We now introduce our first new algorithm `CORE` which aims at generating e-hard solutions by switching from ternary to UC generalization. To that end, `CORE` uses a second incremental SAT instance `dual`, initialized by converting the dual circuit $\neg\Gamma$ to CNF (line 2). Then, instead of applying ternary generalization at line 6, `CORE` queries `dual` under the assumptions D^σ (line 10), that is, the cube induced by σ, to get an unsatisfiable core (cube) U, followed by iteratively minimizing it (line 11). Then, similarly to `TALE`, Q is updated and U is blocked.

Alg. 1 also shows another novel algorithm `ROC`, which applies ternary generalization (line 6), followed by UC generalization with minimization (lines 10 and 11). Despite the overhead, `ROC` often succeeds in generating smaller solutions than `CORE`, which ultimately leads to a reduction in the number of returned solutions. Generalizing further might still be possible, since our UC extraction algorithm might return a local minimum. However, finding the smallest UC would have been extremely costly [24].

Note that the initial `dual` invocation in `CORE` (line 10) is not expected to encounter any conflicts during SAT solving. This is because the assumptions represent a total Boolean input assignment, whose propagation by Boolean Constraint Propagation (BCP) must trigger a conflict with the clause $\neg o$ prior to any decision. This is not the case during the minimization loop in `CORE` (line 11) and any (even the initial) `dual` invocation in `ROC`, where the assumptions might represent a *partial* assignment to the inputs. This, however, is transparent to both the user and the high-level algorithm developer.

Finally, [14] introduced two additional algorithms: `MARS` based on dual-rail generalization, and `DUTY`, which combines between `TALE` and `MARS`. To test the impact of combining dual-rail and UC generalization, our third new algorithm `CARMA` upgrades `MARS` by switching to UC generalization as follows. `CARMA` is similar to `CORE`, but it uses dual-rail encoding and on-the-fly minimization [14] for `plain`, while still using Tseitin encoding for `dual`.

5 Experimental Results

We implemented our new algorithms `CORE`, `ROC` and `CARMA` in `HALL` [14] and compared them to the already implemented `TALE`, `MARS` and `DUTY`, where the default `HALL` uses `IntelSAT` [35] SAT solver for `plain` and `CaDiCaL` [4] for `dual` across all the algorithms (we provide an empirical justification for the default solver selection later in this section). We also ran the duality-based model counter `dualiza` [29] in its two enumeration modes: `sat` and `bdd`.

We used benchmarks from [14] and [26] with some further extensions. All in all, as reported below, we had started with 14 benchmark families, 10 benchmarks in each, and then removed benchmarks solved by none of the solvers in four hours, which left us with 97 benchmarks overall. We transformed each circuit family with multiple outputs to three one-output families (which AllSAT-CT solvers can handle) as follows: we applied either *or* (_or suffix below) or *xor* (_xor suffix below) operator over all the outputs similarly to [14] to create the first two one-output families, and took only the last output to create the last one-output family (_only_last_out suffix below). Below, we list all the benchmark families; the number of instances from each family solved by at least one solver appears in parenthesis:

- **random_control_or (9)**, **random_control_xor (6)** and **random_control_only_last_out (9)** from EPFL benchmark suite [1], used in [14].
- **arithmetic_or (10)**, **arithmetic_xor (1)** and **arithmetic_only_last_out (5)** from EPFL benchmark suite [1], also used in [14].
- **random_circuits_or (9)**, **random_circuits_xor (1)** and **random_circuits_only_last_out (7)**, generated by using aigfuzz [5] as in [14].
- **iscas85_or (10)**, **iscas85_xor (2)** and **iscas85_only_last_out (8)**, used in [26]; this set contains publicly available circuits, including sequential circuits. Since we consider only combinational circuits, we ignored the buffer commands while parsing the files.
- **sta_gen (10)** [14] – Static Timing Analysis (STA) industrial set: a parametrized benchmark family, which encapsulates a variety of real-world STA instances [14]. We removed the two smallest benchmarks resulting in a family of 10 benchmarks.
- **sta_gen_chunks (10)** – another family of STA benchmarks created by parameterizing the *size of each cube*, rather than the *number of inputs (N)*. Given the chunk size K and the constant number $N = 12289$, the formula $F(N, K)$ consists of a disjunction of subformulas $F_1(N, K)$ and $F_2(N, K)$, each comprising a DNF, conjuncted with the selector v_N or $\neg v_N$, respectively. In every DNF, the cubes have K variables and are mutually disjoint. The resulting formula looks as follows, where $j = (N-1)/2$: $F(N) := F_1(N) \vee F_2(N)$, where $F_1(N) := ((v_1 \wedge v_2 \ldots \wedge v_k) \vee \ldots \vee (v_{j-k+1} \wedge \ldots v_{j-1} \wedge v_j)) \wedge v_N$ and $F_2(N) := ((v_{j+1} \wedge v_{j+2} \ldots \wedge v_{j+k}) \vee \ldots \vee (v_{N-K} \wedge \ldots v_{(N-2} \wedge v_{N-1})) \wedge \neg v_N$.

We used Intel® Xeon® machines with 32Gb memory and 3Ghz CPU frequency. We set the timeout to 1 hour and evaluated three criteria. First, *Solved* stands for the number of solved instances. The second one is *PAR-2 score* (similarly to SAT competitions [2]), where every solved benchmark contributes its run-time and every unsolved benchmark contributes twice the timeout. The lower the PAR-2 score, the better. The third criterion is *Quality*: the size (number of cubes) of the DNF, where we compared solvers by their normalized average quality, the *quality* per instance being best-known-DNF-size / current-DNF-size and 0 for unsolved instances (similarly to anytime categories at MaxSAT Evaluations [3]). The quality must be within the interval $[0, 1]$, where the higher the quality, the better.

Table 1 Our results (sorted by PAR-2 scores). The best results in each column are highlighted.

Algorithm	Origin	PAR-2	Solved	Quality
`ROC`	new	24926.845	94	0.966614
`CORE`	new	25938.178	94	0.888548
`CARMA`	new	27073.553	94	0.886870
`TALE`	[14]	92676.526	85	0.568704
`DUTY`	[14]	99447.688	84	0.560391
`MARS`	[14]	198771.013	70	0.412807
`dualiza_sat`	[29]	263810.947	61	0.459656
`dualiza_bdd`	[29]	332953.818	51	0.397921

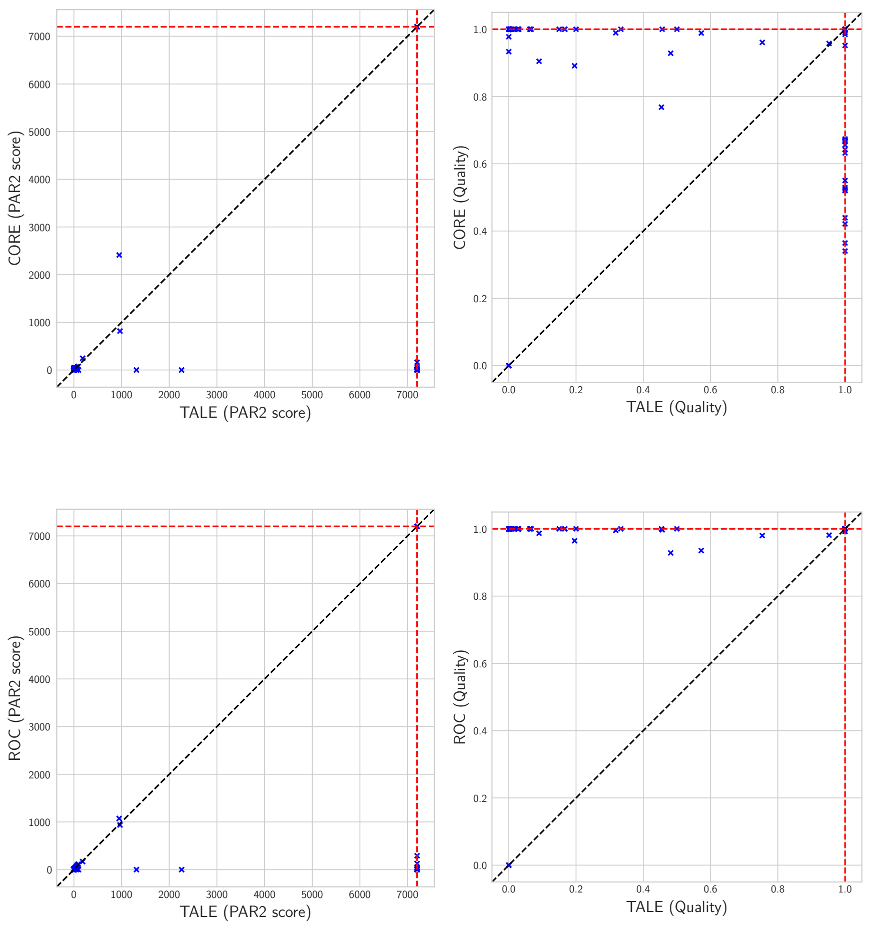


Figure 4 Comparing PAR2 score and quality: `TALE` to `CORE` (top) and `TALE` to `ROC` (bottom)

Table 1 summarizes the main results. `ROC` is the best algorithm by every criterion. It substantially outperforms the previous state-of-the-art (`TALE`), where the gap in quality is especially significant. For an instance-by-instance analysis, consider Fig. 4 starting with its

Table 2 Comparing `TALE` configurations (left) and `CORE` configurations (right).

`TALE` Config.	Solved	Quality
Default `TALE`	85	0.568704
`plain` := `CaDiCaL`	83	0.560356
`plain` := `CryptoMS`	82	0.555761
`plain` := `MergeSat`	80	0.561304
`Backward-TerSim`	72	0.501580

`CORE` Config.	Solved	Quality
Default `CORE`	94	0.888548
`dual` := `CryptoMS`	93	0.846291
`dual` := `IntelSAT`	92	0.872976
`dual` := `MergeSat`	91	0.824571
`No-UC-Minimization`	91	0.707390

upper part, which compares `TALE` to `CORE`. `CORE` is almost always on-par or better in terms of PAR-2 score, but not so in terms of quality. Consider now the lower part of Fig. 4, which compares `TALE` to `ROC`. Unlike `CORE`, `ROC` is either better or on-par with `TALE` in terms of quality on every single instance, whereas `ROC` often yields a substantially better quality. `ROC` is also always on-par or better than `TALE` in terms of PAR-2 score.

Finally, Table. 2 explains our choice of four of `HALL`'s default components. The comparison of `TALE` configurations on the left shows why we set the `plain` SAT solver default to `IntelSAT`, as `IntelSAT` outperforms `CaDiCaL` [4], `MergeSat` [25], and `CryptoMiniSAT` [45] (`CryptoMS` in Table. 2). This result is not surprising as `IntelSAT` was specifically optimized for rapid incremental mostly satisfiable queries [35]. The comparison on the left also explains why we decided against migrating from the default forward ternary generalization to the *backward* one (recall Sect. 3). The right-side table compares `CORE` configurations, supporting the choice of `CaDiCaL` as the default SAT solver for `dual` (notably, in `dual`, unlike in `plain`, the SAT queries are unsatisfiable) and the default inclusion of minimization in UC extraction.

6 Conclusion and Future Work

In this work we substantially improved the state of the art in AllSAT-CT solving in terms of both performance and quality by taking advantage of UC generalization, which can potentially yield solutions that entail the circuit without satisfying it.

Our best-performing algorithm, `ROC`, combines ternary and UC generalization as follows: it iteratively searches for solutions in an `IntelSAT`-based SAT instance `plain`. Then, every solution is generalized using forward ternary generalization, followed by further generalization to its (locally) minimal unsatisfiable core in a `CaDiCaL`-based SAT instance `dual` representing the dual circuit. The generalized solution is then reported to the user and blocked in `plain`. All the algorithms have been implemented in our open-source AllSAT-CT tool `HALL`.

Our results can be relevant for advancing disjoint AllSAT-CT solving and prime implicant enumeration [23]. Furthermore, porting our findings to model checking algorithms such as PDR [6], AVY [51], and CAR [21] could be promising. Notably, while [43] thoroughly compares different generalization approaches within PDR, it surprisingly does not conclude that UC generalization enhances PDR's performance, leaving room for potential improvement.

References

1 Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The EPFL combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.

2 Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2023.

3 Jeremias Berg, Matti Järvisalo, Ruben Martins, and Andreas Niskanen, editors. *MaxSAT Evaluation 2023: Solver and Benchmark Descriptions.* Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2023.

4 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

5 Armin Biere, Keijo Heljanko, and Siert Wieringa. AIGER 1.9 and beyond. Technical Report 11/2, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2011.

6 Aaron R. Bradley. Sat-based model checking without unrolling. In Ranjit Jhala and David A. Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011. `doi:10.1007/978-3-642-18275-4_7`.

7 T. Castell. Computation of prime implicates and prime implicants by a variant of the davis and putnam procedure. In *Proceedings Eighth IEEE International Conference on Tools with Artificial Intelligence*, pages 428–429, 1996. `doi:10.1109/TAI.1996.560739`.

8 Hana Chockler, Alexander Ivrii, Arie Matsliah, Shiri Moran, and Ziv Nevo. Incremental formal verification of hardware. In Per Bjesse and Anna Slobodová, editors, *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, pages 135–143. FMCAD Inc., 2011. URL: `http://dl.acm.org/citation.cfm?id=2157676`.

9 Alejandro Czutro, Ilia Polian, Piet Engelke, Sudhakar M. Reddy, and Bernd Becker. Dynamic compaction in sat-based ATPG. In *Proceedings of the Eighteentgh Asian Test Symposium, ATS 2009, 23-26 November 2009, Taichung, Taiwan*, pages 187–190. IEEE Computer Society, 2009. `doi:10.1109/ATS.2009.31`.

10 David Déharbe, Pascal Fontaine, Daniel Le Berre, and Bertrand Mazure. Computing prime implicants. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 46–52. IEEE, 2013. URL: `https://ieeexplore.ieee.org/document/6679390/`.

11 Imen Ouled Dlala, Saïd Jabbour, Lakhdar Saïs, and Boutheina Ben Yaghlane. A comparative study of SAT-based itemsets mining. In *Research and Development in Intelligent Systems XXXIII - Incorporating Applications and Innovations in Intelligent Systems XXIV. Proceedings of AI-2016.*, pages 37–52, 2016.

12 Niklas Eén, Alan Mishchenko, and Robert Brayton. Efficient implementation of property directed reachability. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*, pages 125–134, 2011.

13 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT, Proceedings*, 2003.

14 Dror Fried, Alexander Nadel, and Yogev Shalmon. AllSAT for combinational circuits. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy*, volume 271 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPIcs.SAT.2023.9`.

15 Alexandra Goultiaeva and Fahiem Bacchus. Exploiting QBF duality on a circuit representation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010. `doi:10.1609/AAAI.V24I1.7548`.

16 James S. Jephson, Robert P. McQuarrie, and Robert E. Vogelsberg. A three-value computer design verification system. *IBM Systems Journal*, 8(3):178–188, 1969.

17 HoonSang Jin, HyoJung Han, and Fabio Somenzi. Efficient conflict analysis for finding all satisfying assignments of a boolean circuit. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3440 of *Lecture Notes in Computer Science*, pages 287–300. Springer, 2005. `doi:10.1007/978-3-540-31980-1_19`.

18 HoonSang Jin and Fabio Somenzi. Prime clauses for fast enumeration of satisfying assignments to boolean circuits. In *Proceedings of the 42nd Design Automation Conference, DAC*, 2005.

19 Alex Kean and George K. Tsiknis. An incremental method for generating prime implicants/impicates. *J. Symb. Comput.*, 9(2):185–206, 1990. `doi:10.1016/S0747-7171(08)80029-6`.

20 Sarfraz Khurshid, Darko Marinov, Ilya Shlyakhter, and Daniel Jackson. A case for efficient solution enumeration. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT*, 2003.

21 Jianwen Li, Shufang Zhu, Yueling Zhang, Geguang Pu, and Moshe Y. Vardi. Safety model checking with complementary approximations. In Sri Parameswaran, editor, *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, CA, USA, November 13-16, 2017*, pages 95–100. IEEE, 2017. `doi:10.1109/ICCAD.2017.8203765`.

22 Nuno P Lopes, Nikolaj Bjørner, Patrice Godefroid, and George Varghese. Network verification in the light of program verification. *MSR, Rep*, 2013.

23 Weilin Luo, Hai Wan, Hongzhen Zhong, Ou Wei, Biqing Fang, and Xiaotong Song. An efficient two-phase method for prime compilation of non-clausal boolean formulae. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021*, pages 1–9. IEEE, 2021. `doi:10.1109/ICCAD51958.2021.9643520`.

24 Inês Lynce and João Marques-Silva. On computing minimum unsatisfiable cores. In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*, 2004. URL: `http://www.satisfiability.org/SAT04/programme/110.pdf`.

25 Norbert Manthey. The mergesat solver. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 387–398. Springer, 2021. `doi:10.1007/978-3-030-80223-3_27`.

26 Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. On CNF conversion for disjoint SAT enumeration. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy*, volume 271 of *LIPIcs*, pages 15:1–15:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPIcs.SAT.2023.15`.

27 E. J. McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956. `doi:10.1002/j.1538-7305.1956.tb03835.x`.

28 Kenneth L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Computer Aided Verification, 14th International Conference, CAV, Proceedings*, 2002.

29 Sibylle Möhle and Armin Biere. Dualizing projected model counting. In *IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI*, pages 702–709, 2018. `doi:10.1109/ICTAI.2018.00111`.

30 Sibylle Möhle, Roberto Sebastiani, and Armin Biere. Four flavors of entailment. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 62–71. Springer, July 2020. `doi:10.1007/978-3-030-51825-7_5`.

31 Sibylle Möhle, Roberto Sebastiani, and Armin Biere. On enumerating short projected models. *CoRR*, abs/2110.12924, October 2021. `arXiv:2110.12924`.

32 Alexander Nadel. Boosting minimal unsatisfiable core extraction. In Roderick Bloem and Natasha Sharygina, editors, *Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23*, pages 221–229. IEEE, 2010. URL: `https://ieeexplore.ieee.org/document/5770953/`.

33 Alexander Nadel. Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In *Formal Methods in Computer Aided Design, FMCAD, Proceedings*, pages 193–202, 2019.

34 Alexander Nadel. Polarity and variable selection heuristics for SAT-based anytime MaxSAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2020.

35 Alexander Nadel. Introducing intel(r) SAT solver. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPIcs.SAT.2022.8`.

36 Alexander Nadel and Vadim Ryvchin. Efficient SAT solving under assumptions. In *Theory and Applications of Satisfiability Testing - SAT, Proceedings*, 2012.

37 Emil L. Post. Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 43, 1921.

38 Alessandro Previti, Alexey Ignatiev, António Morgado, and João Marques-Silva. Prime compilation of non-clausal formulae. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 1980–1988. AAAI Press, 2015. URL: `http://ijcai.org/Abstract/15/281`.

39 W. V. Quine. The problem of simplifying truth functions. *The American Mathematical Monthly*, 59(8):521–531, 1952. `doi:10.1080/00029890.1952.11988183`.

40 Kavita Ravi and Fabio Somenzi. Minimal assignments for bounded model checking. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2988 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2004. `doi:10.1007/978-3-540-24730-2_3`.

41 J. Paul Roth, Willard G. Bouricius, and Peter R. Schneider. Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits. *IEEE Trans. Electron. Comput.*, 16(5):567–580, 1967. `doi:10.1109/PGEC.1967.264743`.

42 Roberto Sebastiani. Are you satisfied by this partial assignment? *CoRR*, abs/2003.04225, February 2020. `arXiv:2003.04225`.

43 Tobias Seufert, Felix Winterer, Christoph Scholl, Karsten Scheibler, Tobias Paxian, and Bernd Becker. Everything you always wanted to know about generalization of proof obligations in PDR. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 42(4):1351–1364, 2023. `doi:10.1109/TCAD.2022.3198260`.

44 James R. Slagle, Chin-Liang Chang, and Richard C. T. Lee. A new algorithm for generating prime implicants. *IEEE Trans. Computers*, 19(4):304–310, 1970. `doi:10.1109/T-C.1970.222917`.

45 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009. `doi:10.1007/978-3-642-02777-2_24`.

46 Robert B. Hitchcock Sr. Timing verification and the timing analysis program. In *Proceedings of the 19th Design Automation Conference, DAC*, 1982.

47 P. Tafertshofer and A. Ganz. Sat based atpg using fast justification and propagation in the implication graph. In *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No.99CH37051)*, pages 139–146, 1999. `doi:10.1109/ICCAD.1999.810638`.

48 Abraham Temesgen Tibebu and Görschwin Fey. Augmenting all solution SAT solving for circuits with structural information. In *21st IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS 2018, Budapest, Hungary, April 25-27, 2018*, pages 117–122. IEEE, 2018. doi:10.1109/DDECS.2018.00028.

49 Takahisa Toda and Takehide Soh. Implementing efficient all solutions SAT solvers. *Journal of Experimental Algorithmics (JEA)*, 2016.

50 Grigori S Tseitin. On the complexity of derivation in propositional calculus. *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pages 466–483, 1983.

51 Yakir Vizel and Arie Gurfinkel. Interpolating property directed reachability. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2014. doi:10.1007/978-3-319-08867-9_17.

52 Yinlei Yu, Pramod Subramanyan, Nestan Tsiskaridze, and Sharad Malik. All-SAT using minimal blocking clauses. *Proceedings of the IEEE International Conference on VLSI Design*, pages 86–91, 2014.