Proofs in Conflict-Driven Theory Combination*

Maria Paola Bonacina
Dipartimento di Informatica
Università degli Studi di Verona
Verona, Italy
mariapaola.bonacina@univr.it

Stéphane Graham-Lengrand
CNRS - INRIA - École Polytechnique
Palaiseau, France
SRI International, Menlo Park, USA
lengrand@lix.polytechnique.fr

Natarajan Shankar
SRI International
Menlo Park, USA
shankar@csl.sri.com

Abstract

Search-based satisfiability procedures try to construct a model of the input formula by simultaneously proposing candidate models and deriving new formulæ implied by the input. When the formulæ are satisfiable, these procedures generate a model as a witness. Dually, it is desirable to have a proof when the formulæ are unsatisfiable. Conflict-driven procedures perform nontrivial inferences only when resolving conflicts between the formulæ and assignments representing the candidate model. CDSAT (*Conflict-Driven SATisfiability*) is a method for conflict-driven reasoning in combinations of theories. It combines solvers for individual theories as *theory modules* within a solver for the union of the theories. In this paper we endow CDSAT with *lemma learning* and *proof generation*. For the latter, we present two techniques. The first one produces *proof objects* in memory: it assumes that all theory modules produce proof objects and it accommodates multiple proof formats. The second technique adapts *the LCF approach to proofs* from interactive theorem proving to conflict-driven SMT-solving and theory combination, by defining a small kernel of reasoning primitives that guarantees that CDSAT proofs are correct by construction.

CCS Concepts • **Theory of computation → Automated reasoning**;

Keywords Proof generation, Lemma learning, Theory combination, Satisfiability modulo assignment

*Part of this research was conducted while the first author was an international observer at the SRI International Computer Science Laboratory, whose support is greatly appreciated. This research was funded in part by the National Science Foundation with grants CCF-1528153 and CNS-0917375, by DARPA under agreement number FA8750-16-C-0043, and by the Università degli Studi di Verona with grant "Ricerca di base 2015." The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, NSF, or the U.S. Government.

CPP'18, January 8–9, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5586-5/18/01…$15.00
https://doi.org/10.1145/3167096

ACM Reference Format:
Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar. 2018. Proofs in Conflict-Driven Theory Combination. In *Proceedings of 7th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP'18).* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3167096

1 Introduction

The satisfiability problem is one of checking if a given formula has a model. Propositional satisfiability (SAT) is the problem of finding an assignment of truth values to variables that satisfies the input propositional formula, typically given as a set of clauses. Many SAT solvers employ a *conflict-driven search strategy*, known as Conflict-Driven Clause Learning (CDCL), in which the solver extends a partial assignment until it satisfies the formula, or a conflict arises because the assignment falsifies the formula. Nontrivial inference steps are performed in response to a conflict to roll back the partial assignment and direct the search elsewhere [27]. This conflict-driven style inspired the design of several solvers for quantifier-free fragments of arithmetic [12, 19, 23–25, 29, 35, 36]. These *conflict-driven theory solvers* decide the satisfiability of sets of literals in the theory. The problem of deciding the satisfiability of sets of clauses in a theory is known as Satisfiability Modulo a Theory (SMT). MCSAT, for *Model Constructing SATisfiability*, is an approach to conflict-driven SMT [14]. It integrates a CDCL-based SAT-solver and a conflict-driven theory solver equipped with an inference system to detect and explain conflicts in the theory. In MCSAT, both Boolean and first-order variables are given assignments in a trail representing the candidate model as a partial assignment. First-order variables are assigned constant symbols representing individuals of the corresponding sort in a model's domain. For example, integer variables are assigned integer constants, and bit vector variables are assigned constant bit vectors. MCSAT allows the introduction of *new* (i.e., non-input) terms in lemmas as long as these terms are drawn from a *finite* basis. The MCSAT approach has been shown to be quite versatile and efficient [21, 22, 38].

Most problems involve more than one theory, and MCSAT is not a combination calculus in general. In prior work, we designed a method called *Conflict-Driven SATisfiability* (CDSAT) to solve in a conflict-driven manner satisfiability problems in a *generic* combination of *disjoint* theories [6, 7]. CDSAT combines *multiple* theory solvers, all or some of

which are conflict-driven, into a conflict-driven solver for the union of the theories. More precisely, CDSAT combines *theory inference systems*, called *theory modules*: the only relevant component of a conflict-driven theory solver is its inference system, since the conflict-driven search is performed by CDSAT for all theories. A non-conflict-driven solver is abstracted into a theory module whose only inference rule invokes the solver to detect unsatisfiability in the theory.

In CDSAT, a common trail contains assignments to both Boolean and first-order *terms*. Some assignments are decisions, that is, guesses, and others are propagated because they are entailed by the formula and prior assignments in the trail. Theory modules work on this trail to detect conflicts with any theory-specific input formula. Conflict resolution rules work to roll back the trail to a prior decision level. The CDSAT transition system is sound, terminating, and complete, relative to the theory modules and a *global* finite basis for all theories. CDSAT generalizes CDCL, MCSAT, and the equality sharing (Nelson-Oppen) method for combination of theories [26, 31]: CDSAT reduces to CDCL, if propositional logic is the only theory, to MCSAT, if the combination features propositional logic and another theory with conflict-driven solver, and to equality sharing, if no theory in the combination has a conflict-driven solver [7].

In this paper we extend CDSAT with *lemma learning* (Section 3) and *proof generation* (Sections 4 and 5). Lemmas are formulæ entailed by the input formula. Learning lemmas from conflicts is a key feature of conflict-driven reasoning engines, because conflict-driven lemmatization immediately thwarts any attempt to repeat a failed search path (e.g., [27]). Proof generating inference systems [33] are important because for many applications a yes/no answer is not enough, and solvers are expected to generate either a satisfying assignment or a proof of unsatisfiability. For instance, proofs are useful for extracting interpolants (see [8] for a survey) to refine abstractions [20] or generate invariants [10].

CDCL-based SAT solvers generate proofs by propositional resolution [39]. Reasoners based on the DPLL($\mathcal{T}$) paradigm [2, 9, 26, 32] generate proofs by propositional resolution with proofs of *theory lemmas* plugged in as external subproofs (see [8], Section 2.4). This structure reflects a hierarchy where propositional logic is at the center and the theories are satellites. In DPLL($\mathcal{T}$) only propositional reasoning is conflict-driven, and non-conflict-driven theory solvers are integrated as black-boxes, so that their proofs also are black-boxes. In CDSAT, propositional logic and the SAT-solver lose their centrality as the sole conflict-driven component. Although propositional logic retains a special role, as all propagated assignments are Boolean, propositional logic is regarded as one of the theories, and all theory modules contribute directly to the proof, including new terms that may appear on the trail whence in the proof. For all these reasons, CDSAT proofs do not fit in the known mold of DPLL($\mathcal{T}$) proofs, and their generation requires machinery that goes beyond propositional resolution plus theory lemmas.

We present two approaches to proof generation in CDSAT. The first one is a *proof-carrying CDSAT transition system* (Section 4), where *proof terms* record the information needed to generate proofs. We describe different ways to turn proof terms into proofs (Section 5), including producing resolution proofs with theory lemmas. The proof objects produced by proof-carrying CDSAT can be directly checked by a verified checker [34] or exported to a proof format that can be verified by proof checkers. Thus, proof-carrying CDSAT can slot into pipelines from proof-search to proof-checking [1, 3, 4], where a minimal amount of proof information (e.g., an unsatisfiable core) might be sufficient for a theorem prover to regenerate a proof in its own format. The second approach consists of specifying a small *kernel* of primitives in LCF style [17, 30], as already explored in the Psyche prover [18], so that building proof objects in memory can be avoided. If CDSAT is implemented on top of this kernel, the LCF type abstraction ensures that an unsat answer is correct by construction, and CDSAT can be used as a trusted external oracle for interactive proof tools.

2 Background on CDSAT

CDSAT is an engine to determine the satisfiability of a quantifier-free formula modulo a combination of theories (SMT) and possibly *modulo an initial assignment* of values to some first-order variables or terms that appear in the formula. We call this generalization of SMT *satisfiability modulo assignment* (SMA). CDSAT combines in a conflict-driven manner inference systems, called *theory modules*, for *component theories* $\mathcal{T}_1, \ldots, \mathcal{T}_n$. The component theories are required to be *disjoint* so that their signatures $\Sigma_1, \ldots, \Sigma_n$ share neither predicate nor function symbols, except equality, which is present in all signatures and for all sorts. In addition to equality, the theories typically share sorts and constant symbols.

Let $\mathcal{T}_\infty$ be the union of $\mathcal{T}_1, \ldots, \mathcal{T}_n$. If $\Sigma_k{=}(S_k, F_k)$, where S_k is the set of sorts and F_k is the set of symbols, for all k, $1 \leq k \leq n$, the signature of $\mathcal{T}_\infty$ is given by $\Sigma_\infty{=}(S_\infty, F_\infty)$, with $S_\infty{=}\bigcup_{k=1}^{n} S_k$ and $F_\infty{=}\bigcup_{k=1}^{n} F_k$. For most problems, one of the component theories is propositional logic, or the theory of Booleans, with logical connectives such as $\wedge$, $\vee$, and $\neg$ as function symbols. For deciding the satisfiability of a set of literals in one or more convex theories (see [8] Definition 14, from [31] page 252), propositional logic is not needed as a component theory. If there is a non-convex theory (e.g., arrays), disjunction and therefore propositional reasoning are needed. Regardless of whether propositional logic is included, all theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$ must be able to interpret the truth values true and false and therefore their signatures are required to have the sort prop = {true, false}, where prop comes from proposition. Then, formulæ are terms of sort prop. We use l for formulæ and t and u for terms of any sort.

CDSAT and theory modules manipulate *assignments*. An input problem $\{l_1, \ldots, l_m\}$, where $l_1, \ldots, l_m$ are formulæ, is seen as an initial assignment $\{l_1 \leftarrow \mathsf{true}, \ldots, l_m \leftarrow \mathsf{true}\}$, even when abbreviated as $\{l_1, \ldots, l_m\}$. Assignments, including initial assignments, are not restricted to assigning truth values; they can assign values of any sort. For example, with propositional logic, a fragment of arithmetic, and the theory of arrays, $(x > 1) \leftarrow \mathsf{false}$, $(x > 1) \vee (y < 0) \leftarrow \mathsf{true}$, $y \leftarrow -1$, $z \leftarrow \sqrt{2}$, $(store(a, i, v) \simeq b) \leftarrow \mathsf{true}$, $select(a, j) \leftarrow 3$, and $(select(a, j) \simeq v) \leftarrow \mathsf{true}$ are all assignments. An assignment that assigns only values of sort prop is called *Boolean*. A singleton assignment that is not Boolean is called a *first-order* assignment, as it assigns a value to a first-order term. If the input problem is an SMT problem, it is given to CDSAT as a Boolean assignment; if it is an SMA problem, it may also include first-order assignments.

First-order assignments are a standard ingredient in the definition of a first-order model. However, in automated reasoning, models need to be represented syntactically. First-order assignments raise the issue of what is a value that can be assigned. The notion of using ground terms to represent individuals in a model's domain and assigning ground terms to first-order variables does not suffice for assignments such as $select(a, j) \leftarrow 3$, where a term gets a value, or $z \leftarrow \sqrt{2}$, where $\sqrt{2}$ may not be a ground term in the given signature of arithmetic. A standard solution is to extend the signature with constant symbols that name all individuals in the domain of the intended model (e.g., the appropriate set of numerals for a fragment of arithmetic). Therefore, we introduce for each theory $\mathcal{T}_k$, $1 \le k \le n$, a *conservative extension* $\mathcal{T}_k^+$ with signature $\Sigma_k^+ = (S_k, F_k^+)$, where F_k^+ adds to F_k a possibly empty set of *new* constant symbols, called $\mathcal{T}_k^+$*-values*, accompanied by new axioms as needed (e.g., $\sqrt{2}$ with $\sqrt{2} \cdot \sqrt{2} \simeq 2$). For numerals, as for true and false, a $\mathcal{T}_k^+$-value is both the domain element and the constant symbol that names it. F_k^+ may be infinite, but it is countable (e.g., using the algebraic reals as real numbers), and only a finite subset of it will be used in any finite derivation. We use symbols such as $\mathfrak{c}$, $\mathfrak{q}$ for $\mathcal{T}_k^+$-values reserving $\mathfrak{b}$ for true or false.

The union of $\mathcal{T}_1^+, \ldots, \mathcal{T}_n^+$ is an extension $\mathcal{T}_\infty^+$ of $\mathcal{T}_\infty$, with signature $\Sigma_\infty^+ = (S_\infty, F_\infty^+)$ for $F_\infty^+ = \bigcup_{k=1}^n F_k^+$. We assume that the extended theories are still disjoint except for the Boolean values true and false. Conservativity of an extension means that $\mathcal{T}_k^+$-unsatisfiability implies $\mathcal{T}_k$-unsatisfiability for formulæ in the original signature Σ_k, so that the problem does not change: if CDSAT detects $\mathcal{T}_k^+$-unsatisfiability, the problem is $\mathcal{T}_k$-unsatisfiable; if the problem is $\mathcal{T}_k$-satisfiable, there is a $\mathcal{T}_k^+$-model that CDSAT can discover.

Definition 2.1 (Assignment)**.** Given a theory $\mathcal{T}$ with signature Σ and extension $\mathcal{T}^+$, a $\mathcal{T}$*-assignment* is a set of pairs $t \leftarrow \mathfrak{c}$, where t is a term and $\mathfrak{c}$ a $\mathcal{T}^+$-value of the same sort.

We use J for generic $\mathcal{T}$-assignments, A for singleton ones, and L or K for Boolean singletons. We abbreviate $l \leftarrow \mathsf{true}$ as l, $l \leftarrow \mathsf{false}$ as $\bar{l}$, and $t \simeq_s u \leftarrow \mathsf{false}$ as $t \not\simeq_s u$, where s is the sort of t and u, that can be omitted when clear from context. The *flip* $\bar{L}$ of L assigns to the same formula the opposite Boolean value. We use H and E for $\mathcal{T}_\infty$-assignments, that we call *assignments* for short. Note that replacing a first-order assignment $t \leftarrow \mathfrak{c}$ by $(t \simeq \mathfrak{c}) \leftarrow \mathsf{true}$ would blur the distinction between terms, on the left of an assignment, and values, on the right of an assignment.

Since assignments are not formulæ, the notion of *endorsement* was introduced to relate assignments and models (see [7], Section 6.3). For the purposes of this paper, it suffices to say that a $\mathcal{T}_\infty^+$-model $\mathcal{M}$ *endorses* an assignment H if (1) for every pair $(t \leftarrow \mathfrak{c})$ in H, $\mathcal{M}$ satisfies the Σ_∞^+-formula $t \simeq \mathfrak{c}$; and (2) $\mathcal{M}$ interprets any two distinct values appearing in H as two distinct elements.[1] In the sequel we write $H \models L$ to state that every $\mathcal{T}_\infty^+$-model endorsing H also endorses L. With a slight abuse of notation and terminology, if no $\mathcal{T}_\infty^+$-model endorses H, we write $H \models \bot$ and say that H is unsatisfiable. Since the objective of this paper is *proof generation*, we are interested in the situation where CDSAT determines that the initial assignment is unsatisfiable.

In CDSAT, a *theory module* is an inference system for theory $\mathcal{T}_k$, $1 \le k \le n$. Its inferences have the form $J \vdash_k L$ where J is a $\mathcal{T}_k$-assignment and L is a Boolean assignment. Theory modules are required to be *sound*: if $J \vdash_k L$ then every model endorsing J endorses L, so that $J \models L$.

A *CDSAT-derivation* transforms a state consisting of a *trail* Γ. A *trail* is a sequence of distinct singleton assignments that are either *decisions*, denoted ${}_?A$, or *justified assignments*, denoted ${}_{H\vdash}A$. A decision is written ${}_?A$ because it is generally a guess. The *justification* H in ${}_{H\vdash}A$ is a set of singleton assignments that appear before A in the trail. A theory inference $J \vdash_k L$ for some k, $1 \le k \le n$, can justify adding ${}_{J\vdash}L$ to the trail. Initial assignments are treated as justified assignments with empty justification, so that a derivation for the input problem $\{A_1, \ldots, A_m\}$ starts with the trail ${}_{\emptyset\vdash}A_1, \ldots, {}_{\emptyset\vdash}A_m$. Justified assignments that are neither due to theory inferences nor initial assignments will be placed on the trail by conflict-solving transitions. Every assignment on the trail has an *identifier*. A trail can be used as an assignment by ignoring order, justifications, and identifiers, and as an *assignment with identifiers* by ignoring order and justifications.

Definition 2.2 (Level)**.** Given a trail Γ with assignments $A_0, \ldots, A_m$, the *level* of a singleton assignment A_i, $0 \le i \le m$, is given by $\mathsf{level}_\Gamma(A_i) = 1 + \max\{\mathsf{level}_\Gamma(A_j) \mid j < i\}$, if A_i is a decision, and $\mathsf{level}_\Gamma(A_i) = \mathsf{level}_\Gamma(H)$, if A_i is a justified assignment with justification H. The *level* of a set of singleton assignments $H \subseteq \Gamma$ is given by $\mathsf{level}_\Gamma(H) = 0$, if $H = \emptyset$, and $\mathsf{level}_\Gamma(H) = \max\{\mathsf{level}_\Gamma(A) \mid A \in H\}$, otherwise.

[1] The notion of endorsement given here corresponds to that of *view-endorsement* (see [7], Definition 7), obtained by combining the notions of *endorsement* for a single theory (see [7], Definition 6) and *theory view* of an assignment (see [7], Definition 2), omitted here for reasons of space.

SEARCH RULES				
Decide	Γ	$\longrightarrow$	$\Gamma, {}_?A$	if A is a $\mathcal{T}_k$-assignment for a term that is $\mathcal{T}_k$-*relevant* for Γ, and A is *acceptable* for the $\mathcal{T}_k$-module and its view of Γ, with $1 \le k \le n$
The next three rules share the conditions: $J \subseteq \Gamma$, $(J \vdash_k L)$, and $L \notin \Gamma$, for some k, $1 \le k \le n$.				
Deduce	Γ	$\longrightarrow$	$\Gamma, {}_{J\vdash}L$	if $\overline{L} \notin \Gamma$ and L is in $\mathcal{B}$
Fail	Γ	$\longrightarrow$	unsat	if $\overline{L} \in \Gamma$ and $\text{level}_\Gamma(J \cup \{\overline{L}\}) = 0$
ConflictSolve	Γ	$\longrightarrow$	Γ'	if $\overline{L} \in \Gamma$, $\text{level}_\Gamma(J \cup \{\overline{L}\}) > 0$, and $\langle \Gamma; J \cup \{\overline{L}\}\rangle \Longrightarrow^* \Gamma'$
CONFLICT RESOLUTION RULES				
Undo	$\langle \Gamma; E \uplus \{A\}\rangle$	$\Longrightarrow$	$\Gamma^{\le m-1}$	if A is a first-order decision of level $m > \text{level}_\Gamma(E)$
Resolve	$\langle \Gamma; E \uplus \{A\}\rangle$	$\Longrightarrow$	$\langle \Gamma; E \cup H\rangle$	if ${}_{H\vdash}A$ is in Γ and H does not contain a first-order decision A' whose level is $\text{level}_\Gamma(E \uplus \{A\})$
UndoDecide	$\langle \Gamma; E \uplus \{L\}\rangle$	$\Longrightarrow$	$\Gamma^{\le m-1}, {}_?\overline{L}$	if ${}_{H\vdash}L$ is in Γ, $m = \text{level}_\Gamma(E) = \text{level}_\Gamma(L)$ and H contains a first-order decision A' of level m
LearnBackjump	$\langle \Gamma; E \uplus H\rangle$	$\Longrightarrow$	$\Gamma^{\le m}, {}_{E\vdash}L$	if L is a clausal form of H and is in $\mathcal{B}$, $L \notin \Gamma$, $\overline{L} \notin \Gamma$, and $\text{level}_\Gamma(E) \le m < \text{level}_\Gamma(H)$

Figure 1. The CDSAT transition system with lemma learning

An assignment of level m may be added after one of level r, $r > m$; $\Gamma^{\le m}$ denotes the restriction of Γ to its elements of level at most m. The rules of the CDSAT transition system comprise *search rules*, whose application is denoted by $\longrightarrow$, and *conflict resolution rules*, whose application is denoted by $\Longrightarrow$, with transitive closure $\Longrightarrow^*$ (see Fig. 1). The CDSAT rules may place on the trail assignments for *new* terms, meaning terms that do not appear in the input. For termination, these terms must come from a *finite* set $\mathcal{B}$ of terms, called *global basis* (cf. [7], Section 6.2). $\mathcal{B}$ is a parameter of the CDSAT transition system, whose instantiation depends on the input problem. The distinction between terms and $\mathcal{T}^+$-values plays a role here, as terms come from $\mathcal{B}$ and $\mathcal{T}^+$-values come from F^+_∞: $\mathcal{B}$ is finite and fixed once the input is given and throughout the derivation, whereas F^+_∞ may be an infinite supply out of which a derivation uses a finite subset that is not fixed prior to the start of the derivation. An assignment H is in $\mathcal{B}$ if $t \in \mathcal{B}$ for all t such that $(t \leftarrow c) \in H$.

Rule Decide adds to the trail Γ a singleton assignment that is a decision, provided the term is *relevant* for a theory and the assignment *acceptable* for the theory module. Relevance (see [7], Definition 4) organizes the division of labor among theories. Acceptability (see [7], Definition 3) prevents adding an assignment that is already in Γ or that is so immediately inconsistent with Γ that it would have to be withdrawn without learning anything, such as adding L if $\overline{L}$ is in Γ or $1 \leftarrow 3$ in arithmetic with 1 a constant symbol in F_∞. Acceptability and relevance also imply that A is in $\mathcal{B}$, which is relevant for termination [6, 7]. Rule Deduce extends Γ with a Boolean singleton assignment justified by a theory inference $J \vdash_k L$ from assignments J already in Γ. The system proceeds with decisions and deductions until a conflict arises, as $J \vdash_k L$, for some k, $1 \le k \le n$, and $\overline{L} \in \Gamma$: the assignment $J \cup \{\overline{L}\}$ is a *conflict*, because it is unsatisfiable. If the conflict is at level 0, rule Fail reports unsatisfiability. If the conflict is at a level greater than 0, rule ConflictSolve passes the control to the conflict resolution rules, and resumes the search from the trail Γ' resulting from solving the conflict.

The conflict resolution rules operate on a trail and a *conflict* that is an unsatisfiable set of assignments from the trail. These rules transform the conflict until either it is solved or it surfaces at level 0 so that Fail fires. The rules use the notation $\uplus$ for disjoint set union. Rule Undo applies if the conflict includes a first-order decision A whose level m is greater than that of any other element in the conflict: the effect of the rule is to exit the conflict by jumping back to level $m-1$, removing A and all its consequences from the trail. While it may seem that the resulting trail was already encountered, jeopardising termination, this is not the case [6]. Acceptability ensures that A did not cause an inconsistency right away when it was added to the trail. If A is now part of a conflict, it means that some justified assignment L was added to the trail *after* decision A even if $\text{level}_\Gamma(L) < m$ (*late propagation*). Thus, $\Gamma^{\le m-1}$ is new because it contains L.

Rule Resolve *explains* the conflict, by replacing a justified assignment A with its justification H, unless H contains a first-order decision A' whose level m is that of the current conflict. In that case, using Resolve to replace A by its justification H is forbidden, and it can be shown that A must be a Boolean assignment L also of level m so that another

rule applies [6]. Indeed, if an assignment other than L in the conflict has level m, rule UndoDecide undoes A' and replaces it by a Boolean decision on $\overline{L}$. If *only one* of the assignments in the conflict has the same level as the conflict, then the conflict is solved by a Backjump rule [6, 7], replaced here with LearnBackjump to add *learning* as illustrated in Section 3.

The CDSAT transition system is non-deterministic as the rules leave room for heuristic choices. Thus, there are multiple CDSAT-derivations from a given input problem. In order to get a *CDSAT procedure* that yields a unique CDSAT-derivation from a given input, the transition rules must be coupled with a *search plan* that drives their application.

3 CDSAT with Lemma Learning

In order to enrich the CDSAT transition system with *lemma learning*, we turn assignments into clauses that the system can learn. First, we do this only for assignments that partake in conflicts, because in a conflict-driven system learning is also conflict-driven [27]. Second, only Boolean assignments can be turned into clauses. Third, in order to write non-unit clauses, we need disjunction. If propositional logic *is not* one of the combined theories, disjunction is not in the combined signature, and the only assignments that can yield clauses are singleton Boolean assignments that yield unit clauses. If propositional logic *is* one of the combined theories, any Boolean assignment can yield a clause.

Suppose that $E \uplus H$ is a conflict, where H contains only Boolean assignments. This means that $E \uplus H \models \bot$. If H is a singleton L, we have $E \uplus \{L\} \models \bot$, hence $E \models \overline{L}$, and the flip $\overline{L}$ is the clause associated to assignment L in the conflict. If H is not a singleton, it can be rewritten as the singleton

$$((\bigwedge_{(l \leftarrow \mathsf{true}) \in H} l) \wedge (\bigwedge_{(l \leftarrow \mathsf{false}) \in H} \neg l)) \leftarrow \mathsf{true}$$

that can be flipped into

$$((\bigwedge_{(l \leftarrow \mathsf{true}) \in H} l) \wedge (\bigwedge_{(l \leftarrow \mathsf{false}) \in H} \neg l)) \leftarrow \mathsf{false}.$$

In order to get a clause, the above assignment can be rewritten in the equivalent form

$$((\bigvee_{(l \leftarrow \mathsf{true}) \in H} \neg l) \vee (\bigvee_{(l \leftarrow \mathsf{false}) \in H} l)) \leftarrow \mathsf{true}$$

leading to the next definition.

Definition 3.1 (Clausal form of an assignment in a conflict)**.** Given a conflict $E \uplus H$, where H is a Boolean assignment, the singleton Boolean assignments

$$((\bigvee_{(l \leftarrow \mathsf{true}) \in H} \neg l) \vee (\bigvee_{(l \leftarrow \mathsf{false}) \in H} l)) \leftarrow \mathsf{true}$$
$$((\bigwedge_{(l \leftarrow \mathsf{true}) \in H} l) \wedge (\bigwedge_{(l \leftarrow \mathsf{false}) \in H} \neg l)) \leftarrow \mathsf{false}$$

are *clausal forms* of H.

We now describe the new rule LearnBackjump, listed last in Fig. 1. Rule LearnBackjump is quite versatile, since

1. It allows CDSAT to perform *learning and backjumping*;
2. It allows CDSAT to perform *learning and restart*;
3. It covers as a special case the Backjump rule of the basic version of CDSAT [7], adding the capability to *learn assertion clauses.*

Input problem H_0 including:	$(\neg l_4 \vee l_5)$, $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
Initial trail Γ_0 including:	${}_{\emptyset\vdash}(\neg l_4 \vee l_5)$, ${}_{\emptyset\vdash}(\neg l_2 \vee \neg l_4 \vee \neg l_5)$
Extending Γ_0 into $\Gamma = \Gamma_0$,	${}_{?}A_1$, ${}_{?}l_2$, ${}_{?}A_3$, ${}_{?}l_4$, ${}_{(\neg l_4 \vee l_5),\, l_4\vdash} l_5$
	(involving unrelated decisions A_1 and A_3)
First conflict:	$\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, l_5 \rangle$
Applying Resolve to l_5:	$\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, (\neg l_4 \vee l_5) \rangle$

Figure 2. Propositional extract from a CDSAT derivation

We consider these three features in this order.

Learning and backjumping is the generic behavior of rule LearnBackjump. This rule singles out a Boolean subset H of the conflict $E \uplus H$, such that $\mathsf{level}_\Gamma(H) > \mathsf{level}_\Gamma(E)$. Then, it solves the conflict by jumping back to a level m, such that $\mathsf{level}_\Gamma(E) \leq m < \mathsf{level}_\Gamma(H)$, and learning a clausal form L of H. The system learns L by adding to the trail the justified assignment ${}_{E\vdash}L$, since $E \uplus H \models \bot$ implies $E \models L$, as L is a clausal form of H. The clausal form L may assign a Boolean value to a new term, and therefore it must be an element of $\mathcal{B}$. Note that H does not necessarily contain all the Boolean assignments in the conflict: the choices of the Boolean subset H and the destination level m are left to the search plan.

Example 3.2. Consider the conflict on the last line of Fig. 2. Assume that LearnBackjump is applied with $H = \{l_2, l_4\}$, $E = \{(\neg l_2 \vee \neg l_4 \vee \neg l_5), (\neg l_4 \vee l_5)\}$, where $((\neg l_2 \vee \neg l_4) \leftarrow \mathsf{true})$ is a clausal form of H. We have $\mathsf{level}_\Gamma(H) = 4$ and $\mathsf{level}_\Gamma(E) = 0$, so that any destination level m such that $0 \leq m < 4$ can be picked. A standard choice for m would be the second highest level in the conflict, namely $m = 2$, in which case the LearnBackjump step jumps over decision A_3 and yields

$$\Gamma_0,\ {}_{?}A_1,\ {}_{?}l_2,\ {}_{(\neg l_2 \vee \neg l_4 \vee \neg l_5),\, (\neg l_4 \vee l_5)\vdash}(\neg l_2 \vee \neg l_4).$$

The derivation continues from level 2 with the learned lemma $\neg l_2 \vee \neg l_4$ added to level 0.

We consider next *learning and restart*. It is common to restart after learning a clause, and search plans with aggressive restart proved successful in SAT solving. LearnBackjump makes this kind of search plan possible in CDSAT. Assume that the destination level m is chosen to be the *smallest*, that is, $m = \mathsf{level}_\Gamma(E)$. If $\mathsf{level}_\Gamma(E)$ is 0, the shape of the trail after LearnBackjump is $\Gamma^{\leq 0}, {}_{E\vdash}L$, which means LearnBackjump performs a *restart* and adds ${}_{E\vdash}L$ to level 0.

Example 3.3. Consider the same LearnBackjump step as in Example 3.2 except that the destination level is $m = 0$. We get

$$\Gamma_0,\ {}_{(\neg l_2 \vee \neg l_4 \vee \neg l_5),\, (\neg l_4 \vee l_5)\vdash}(\neg l_2 \vee \neg l_4)$$

as all decisions are undone and the derivation restarts with the learned lemma $\neg l_2 \vee \neg l_4$ added to level 0.

In the rest of this section we analyze how LearnBackjump subsumes Backjump [7] and allows CDSAT to learn assertion clauses. The Backjump rule applies when CDSAT reaches a conflict state $\langle \Gamma; E \uplus \{L\} \rangle$, where $\mathsf{level}_\Gamma(L) > m$, with

$m = \mathsf{level}_\Gamma(E)$. Backjump solves such a conflict by producing $\Gamma^{\leq m}, {}_{E}\vdash \overline{L}$. In words, it jumps back to level m and adds to the trail the justified assignment ${}_{E}\vdash \overline{L}$, because $E \uplus \{L\} \models \bot$ yields $E \models \overline{L}$. Rule LearnBackjump can behave in the same way by taking as H a singleton L. Indeed, $\overline{L}$ is a clausal form of any singleton Boolean assignment L in the conflict. However, even when it reproduces Backjump, LearnBackjump allows a choice of destination level: Backjump goes back to level $m = \mathsf{level}_\Gamma(E)$, while LearnBackjump allows the system to go back to any level m such that $\mathsf{level}_\Gamma(E) \leq m < \mathsf{level}_\Gamma(L)$.

Example 3.4. The conflict on the last line of Fig. 2 contains an assignment, namely l_4, whose level is greater than that of the rest of the assignment, as $\mathsf{level}_\Gamma(l_4) = 4 > \mathsf{level}_\Gamma(E) = 2$ where $E = \{(\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, (\neg l_4 \vee l_5)\}$. Thus, Backjump could apply and we can see how LearnBackjump mimics it. A LearnBackjump step with $H = \{l_4\}$ and $m = 2$ yields

$$\Gamma_0,\ {}_?A_1,\ {}_?l_2,\ {}_{(\neg l_2 \vee \neg l_4 \vee \neg l_5),\, l_2,\, (\neg l_4 \vee l_5)}\vdash \overline{l_4}.$$

Alternatively, if $m = 3$, LearnBackjump yields

$$\Gamma_0,\ {}_?A_1,\ {}_?l_2,\ {}_?A_3,\ {}_{(\neg l_2 \vee \neg l_4 \vee \neg l_5),\, l_2,\, (\neg l_4 \vee l_5)}\vdash \overline{l_4}.$$

The side-condition $L \notin \Gamma$ of LearnBackjump prevents adding to the trail clauses that are already there.

Example 3.5. LearnBackjump can be applied also to the first conflict in Fig. 2, namely $\langle \Gamma; (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, l_5 \rangle$. Consider a step with $E = \{\neg l_2 \vee \neg l_4 \vee \neg l_5\}$ and $H = \{l_2, l_4, l_5\}$, so that $\mathsf{level}_\Gamma(H) = 4$ and $\mathsf{level}_\Gamma(E) = 0$. Regardless of the choice of destination level m, $0 \leq m < 4$, a clausal form of H is redundant since clause $\neg l_2 \vee \neg l_4 \vee \neg l_5$ is already on the trail and LearnBackjump does not add it.

Unlike Backjump, LearnBackjump does *not* require the occurrence in the conflict of a singleton assignment L of level greater than the rest of the conflict.

Example 3.6. In the first conflict in Fig. 2 both l_4 and l_5 have level 4. If we apply LearnBackjump with $H = \{l_4, l_5\}$, $E = \{(\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2\}$, $\mathsf{level}_\Gamma(H) = 4$, $\mathsf{level}_\Gamma(E) = 2$, and destination level $m = 2$, the resulting trail is

$$\Gamma_0,\ {}_?A_1,\ {}_?l_2,\ {}_{E}\vdash \neg l_4 \vee \neg l_5$$

where $(\neg l_4 \vee \neg l_5) \leftarrow \mathsf{true}$ on level 2 is a clausal form of H.

The last conflict clause generated prior to backjumping is called *backjump clause*: the system learns this clause and jumps back to a prior level in such a way to undo at least one decision and satisfy the learned clause by placing on the trail one of its literals. The *First Unique Implication Point* (1UIP) heuristic [28] picks as the backjump clause the *first* generated assertion clause, where an *assertion clause* is a conflict clause where only one of its literals, say l, is falsified at the current level. The Backjump rule of CDSAT generalizes this principle, because it applies when the conflict contains a singleton assignment L of level greater than the rest of the conflict (not necessarily the current one). However, Backjump places $\overline{L}$ on the trail without learning an assertion clause. When a CDSAT conflict contains a singleton assignment L of level greater than the rest of the conflict, it means it is possible to extract from the conflict an assertion clause. Thus, with rule LearnBackjump, CDSAT can learn assertion clauses, and a CDSAT search plan may choose to restrict the application of this rule to 1UIP conflict clauses.

Let $\kappa = l_1 \vee \cdots \vee l_q$ be an assertion clause that is in the global basis $\mathcal{B}$. Let l_q be the literal of κ made false by the current level and L the assignment that makes l_q false. In order to learn κ, it suffices to identify the Boolean part $H = H' \uplus \{L\}$ of the conflict that makes κ false: for all i, $1 \leq i \leq q$, $(l_i \leftarrow \mathsf{true}) \in H$ if and only if $\neg l_i \in \kappa$ and $(l_i \leftarrow \mathsf{false}) \in H$ if and only if $l_i \in \kappa$. By Definition 3.1, the assignment $K = (\kappa \leftarrow \mathsf{true})$ is a clausal form of H. Let E be the rest of the conflict. Then the system applies LearnBackjump with destination level $m = \mathsf{level}_\Gamma(E \uplus H')$, which means $m = \mathsf{level}_\Gamma(E)$ if $\mathsf{level}_\Gamma(E) > \mathsf{level}_\Gamma(H')$, $m = \mathsf{level}_\Gamma(H')$ if $\mathsf{level}_\Gamma(H') > \mathsf{level}_\Gamma(E)$, and $m = \mathsf{level}_\Gamma(H') = \mathsf{level}_\Gamma(E)$ if $\mathsf{level}_\Gamma(H') = \mathsf{level}_\Gamma(E)$. This choice satisfies the condition $\mathsf{level}_\Gamma(E) \leq m < \mathsf{level}_\Gamma(H)$, because $\mathsf{level}_\Gamma(E) \leq \mathsf{level}_\Gamma(E \uplus H') < \mathsf{level}_\Gamma(H) = \mathsf{level}_\Gamma(L)$. This LearnBackjump step yields the trail

$$\Gamma^{\leq m}, {}_{E}\vdash K,$$

and κ is learned. The theory module for propositional logic features inference rules for unit propagation (cf. [7], Section 4.1) that allow the inference:

$$\{K\} \uplus H' \vdash_{\mathsf{Bool}} \overline{L}. \tag{1}$$

Indeed, K is $(l_1 \vee \ldots \vee l_{q-1} \vee l_q) \leftarrow \mathsf{true}$, and H' makes $l_1, \ldots, l_{q-1}$ false, so that unit propagation infers l_q. Since L makes l_q false, $\overline{L}$ makes l_q true. Because the destination level m of the LearnBackjump step was chosen in such a way that $m \geq \mathsf{level}_\Gamma(H')$, the premises K, H' of inference (1) are all on the trail $\Gamma^{\leq m}, {}_{E}\vdash K$. Furthermore, literal l_q is in $\mathcal{B}$, since L was on the trail. Thus, all conditions are met to apply a Deduce step with inference (1). The resulting trail is

$$\Gamma^{\leq m}, {}_{E}\vdash K, {}_{\{K\} \uplus H'}\vdash \overline{L}$$

which is similar to the $\Gamma^{\leq m}, {}_{E \uplus H'}\vdash \overline{L}$ produced by Backjump, except for the learned clause K. The advantage is that K can be reused in future branches of the search. The level of ${}_{E}\vdash K$ is $\mathsf{level}_\Gamma(E)$: the smaller $\mathsf{level}_\Gamma(E)$ is, the longer K may remain on the trail and be used for inferences.

Example 3.7. Continuing Example 3.2 from

$$\Gamma_0,\ {}_?A_1,\ {}_?l_2,\ {}_{(\neg l_2 \vee \neg l_4 \vee \neg l_5),\, (\neg l_4 \vee l_5)}\vdash (\neg l_2 \vee \neg l_4),$$

rule Deduce with inference (1) generates

$$\Gamma_0,\ {}_?A_1,\ {}_?l_2,\ {}_{(\neg l_2 \vee \neg l_4 \vee \neg l_5),\, (\neg l_4 \vee l_5)}\vdash (\neg l_2 \vee \neg l_4),\ {}_{(\neg l_2 \vee \neg l_4),\, l_2}\vdash \overline{l_4}.$$

CDSAT with lemma learning inherits the *soundness*, *completeness*, and *termination* properties of CDSAT, under suitable assumptions on theory modules and the input-dependent

global basis $\mathcal{B}$ (see [7], Section 6, and [6], Sections 8 and 9). Intuitively, $\mathcal{B}$ needs to be finite for termination, and sufficiently large for completeness. Informally speaking, the latter means that $\mathcal{B}$ contains all input terms and all terms that the theory modules may need, as defined by n local basis for the combined theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$. Such a global basis is called *stable*. The theory modules need to be *complete*, which means that when none of them can extend the trail, the assignment on the trail is endorsed by a $\mathcal{T}_\infty^+$-model.

Theorem 3.8. *CDSAT with lemma learning is* sound *(if it derives unsat, the input problem is unsatisfiable),* complete *(if the theory modules are complete, and the global basis is stable, whenever the derivation reaches a state other than unsat such that no transition rule applies, there exists a model that endorses the input problem), and* terminating *(every derivation with a finite global basis is guaranteed to terminate).*

We leave for future work the performance analysis of different ways to apply LearnBackjump, focusing next on proof generation.

4 Proof Object Production in CDSAT

In this section we describe the first technique to endow CDSAT with *proof generation*. This technique enables the system to build proof objects in memory. When a CDSAT derivation terminates by firing rule Fail and detecting unsatisfiability, the system engages in an activity called *proof reconstruction* [5], which consists of extracting a proof object from the final state of the derivation. For this, we need to instrument the transition system in such a way that the final state contains enough information to reconstruct a proof object.

4.1 Theory-Specific Proofs

Because CDSAT combines theory modules, proof object production in CDSAT requires proof object production in every theory module. Therefore, we assume that each theory module is equipped with a *proof annotation system* that *annotates* theory inferences with *theory proofs*:

$$J \vdash_k j_k : L$$

means that the module for theory $\mathcal{T}_k$ infers L from J with theory proof j_k. A theory proof refers to L and the individual assignments in J by means of *identifiers*: in $J \vdash_k j_k : L$, the premise J is a *$\mathcal{T}_k$-assignment with identifiers*, that is, a set of triples ${}^a(t \leftarrow \mathfrak{c})$, where a is the *identifier* of the singleton $\mathcal{T}_k$-assignment $t \leftarrow \mathfrak{c}$.

For instance, inference (1) from Section 3, representing Unit Propagation (UP) in the theory module for propositional logic, can be annotated with a theory proof denoted $\mathsf{UP}(a, \{a_1, \ldots, a_n\})$, as follows:

$$\{{}^a K\} \uplus H' \vdash_{\mathsf{Bool}} \mathsf{UP}(a, \{a_1, \ldots, a_n\}) : \overline{L}$$

where $a_1, \ldots, a_n$ are the identifiers of the assignments in H'. For Linear Rational Arithmetic (LRA) annotated inferences include instances of Fourier-Motzkin resolution

$$\frac{}{{}^{a_1}(t_1 \leftarrow \mathfrak{c}_1), {}^{a_2}(t_2 \leftarrow \mathfrak{c}_2) \vdash \mathsf{eq}(a_1, a_2) : t_1 \simeq t_2} \text{ if } \mathfrak{c}_1 \text{ and } \mathfrak{c}_2 \text{ are the same } \mathcal{T}^+\text{-value of sort } s$$

$$\frac{}{{}^{a_1}(t_1 \leftarrow \mathfrak{c}_1), {}^{a_2}(t_2 \leftarrow \mathfrak{c}_2) \vdash \mathsf{neq}(a_1, a_2) : t_1 \not\simeq t_2} \text{ if } \mathfrak{c}_1 \text{ and } \mathfrak{c}_2 \text{ are distinct } \mathcal{T}^+\text{-values of sort } s$$

$$\frac{}{\vdash \mathsf{refl} : t \simeq t}$$

$$\frac{}{{}^a(t_1 \simeq t_2) \vdash \mathsf{sym}(a) : t_2 \simeq t_1}$$

$$\frac{}{{}^{a_1}(t_1 \simeq t_2), {}^{a_2}(t_2 \simeq t_3) \vdash \mathsf{trans}(a_1, a_2) : t_1 \simeq t_3}$$

Figure 3. Equality inference rules for terms of sort s

$$^{a_1}(e_1 \leq t), {}^{a_2}(t \leq e_2) \vdash_{\mathsf{LRA}} \mathsf{FM}(a_1, a_2) : (e_1 \leq e_2)$$

where t, e_1, and e_2 are rational terms, and evaluation

$$^{a_1}(x_1 \leftarrow \mathfrak{q}_1), \ldots, {}^{a_m}(x_m \leftarrow \mathfrak{q}_m) \vdash_{\mathsf{LRA}} \mathsf{eval}(\{a_1, \ldots, a_m\}) : l \leftarrow \mathfrak{b}$$

where formula l evaluates to the Boolean $\mathfrak{b}$ given the first-order assignments $x_1 \leftarrow \mathfrak{q}_1, \ldots, x_m \leftarrow \mathfrak{q}_m$ for rational variables. For Equality with Uninterpreted Functions (EUF), if J is ${}^{a_1}(t_1 \simeq u_1), \ldots, {}^{a_m}(t_m \simeq u_m)$, a congruence inference is:

$$J \vdash_{\mathsf{EUF}} \mathsf{Cong}(a_1, \ldots, a_m) : f(t_1, \ldots, t_m) \simeq f(u_1, \ldots, u_m).$$

Theory modules to be combined in CDSAT are required to contain basic inference rules for equality. This means that theory modules must provide theory proofs

$$\mathsf{refl}, \mathsf{sym}(a), \mathsf{trans}(a_1, a_2), \mathsf{eq}(a_1, a_2), \mathsf{neq}(a_1, a_2),$$

that annotate equality inferences, as illustrated in Fig. 3.

If an assignment appears on the trail, its identifier in any theory proof is the same as its identifier on the trail. For example, when a Deduce step uses a theory inference $J \vdash_k j_k : L$, the assignments in J appear on the trail Γ, and their identifiers in j_k are the same as in Γ. However, since identifiers are mere names, theory proof annotations are *stable under permutations of identifiers*: any permutation π of identifiers transforms a theory proof j_k into a theory proof $\pi(j_k)$, such that, if ${}^{a_1}(t_1 \leftarrow \mathfrak{c}_1), \ldots, {}^{a_m}(t_m \leftarrow \mathfrak{c}_m) \vdash_k j_k : L$ then

$$^{\pi(a_1)}(t_1 \leftarrow \mathfrak{c}_1), \ldots, {}^{\pi(a_m)}(t_m \leftarrow \mathfrak{c}_m) \vdash_k \pi(j_k) : L.$$

Example 4.1. If we have

$$^1(x \leftarrow \mathfrak{c}_1), {}^4(y \leftarrow \mathfrak{c}_1) \vdash_k eq(1, 4) : f(x) \simeq f(y),$$

$\pi(1) = 4$, and $\pi(4) = 1$, then we also have

$$^4(x \leftarrow \mathfrak{c}_1), {}^1(y \leftarrow \mathfrak{c}_1) \vdash_k eq(4, 1) : f(x) \simeq f(y).$$

The assumption that every theory module has a proof annotation system is no restriction, as the proof annotation system can be a trivial one that uses a dummy theory proof for all theory inferences. The resulting theory proofs convey no information, which is acceptable if there are no requirements on the information they should offer.

4.2 Proof Terms

In order to empower CDSAT to generate proof objects, we instrument the CDSAT transition system to keep track of *provability invariants*. We call the outcome *proof-carrying CDSAT*. The provability invariants of a transition system ensure that the transitions do not change the problem. If

$$\frac{A \text{ is initial}}{\emptyset \vdash \mathsf{in}(A) : A} \qquad \frac{J \vdash_k j_k : L}{J \vdash j_k : L} \qquad \frac{E \uplus H \vdash c : \bot}{E \vdash \mathsf{lem}(H.c) : L}\ L \text{ is a clausal form of } H \qquad \frac{J \vdash_k j_k : L}{J \cup \{{}^a\overline{L}\} \vdash \mathsf{cfl}(j_k, a) : \bot} \qquad \frac{H \vdash j : A \quad E, {}^aA \vdash c : \bot}{E \cup H \vdash \mathsf{res}(j, {}^aA.c) : \bot}$$

Figure 4. Proof system for the CDSAT proof terms

the transition rules preserve such invariants, they are sound. The provability invariants of CDSAT are:

1. For every assignment $_{H\vdash}A$ on the trail, $H \models A$;
2. For every conflict state $\langle \Gamma; E\rangle$, $E \models \bot$.

Proof-carrying CDSAT keeps track of provability invariants by means of a *trace* of *proof terms*. We distinguish between *deduction proof terms* and *conflict proof terms*, intuitively related to Invariants (1) and (2), respectively. A deduction proof term records why an assignment $_{H\vdash}A$ is on the trail, while a conflict proof term records why the second component of a conflict state $\langle \Gamma; E\rangle$ is a conflict.

Definition 4.2 (CDSAT proof terms)**.** A *CDSAT proof term* is either a *deduction proof term*

$$j ::= \ \mathsf{in}(A) \,\big|\, j_k \,\big|\, \mathsf{lem}(H.c)$$

or a *conflict proof term*

$$c ::= \ \mathsf{cfl}(j_k, a) \,\big|\, \mathsf{res}(j, {}^aA.c)$$

where in, lem, cfl, and res are the CDSAT *proof constructors*, j_k ranges over theory proofs for $\mathcal{T}_k$, $1 \le k \le n$, A is a singleton assignment, and H is a Boolean assignment with identifiers; $\mathsf{res}(j, {}^aA.c)$ binds a in c, and $\mathsf{lem}(H.c)$ binds the identifiers of H in c, as represented by the dot.

The proof system of Fig. 4 helps understanding the invariants that proof terms keep track of. The first three rules establish the derivability of judgments of the form $H \vdash j : A$, witnessing invariant $H \models A$. Proof term $\mathsf{in}(A)$, where in stands for *initial*, witnesses the fact that an initial assignment A holds. The second rule shows that a theory proof j_k can be *coerced* into a CDSAT deduction proof term. The rule for $\mathsf{lem}(H.c)$, where lem stands for *lemma*, shows that, whenever there is a conflict involving a Boolean assignment H, a clausal form of H is entailed by the rest of the conflict and therefore it is a lemma. The proof term carries H to indicate which part of the conflict is turned into a lemma. Since identifiers of assignments in H may occur in c, such occurrences are bound in $\mathsf{lem}(H.c)$.

The last two rules establish the derivability of judgments of the form $E \vdash c : \bot$, witnessing invariant $E \models \bot$. Proof term $\mathsf{cfl}(j_k, a)$, where cfl stands for *conflict*, witnesses the conflict between the conclusion L of a theory inference and its flip $\overline{L}$ (having identifier a). Proof term $\mathsf{res}(j, {}^aA.c)$, where res stands for *resolve*, is the only construct that combines two subproofs, connecting the conclusion A of the left premise with the hypothesis ${}^aA$ of the right premise. Any occurrence of a in c is bound in $\mathsf{res}(j, {}^aA.c)$. Using the soundness of theory inferences, a structural induction on proof terms turns the intuition of witnessing invariants into the following theorem.

Theorem 4.3. *If $H \vdash j : A$, then $H \models A$; if $E \vdash c : \bot$, then $E \models \bot$.*

This choice of proof terms and proof system allows us to associate to every rule of the CDSAT transition system the simplest trace of how the rule transforms the relevant provability invariant, leading us to the next section.

4.3 Proof-Carrying CDSAT

The *proof-carrying CDSAT* transition system is given in Fig. 5. While a decision $_?A$ does not carry a proof term, a justified assignment $_{H\vdash j:}A$ carries a deduction proof term j such that $H \vdash j : A$. Initial assignments take the form $_{\emptyset\vdash \mathsf{in}(A):}A$ where the deduction proof term presents the initial assignment as a premise of the proof. If a theory inference $J \vdash_k L$, $1 \le k \le n$, supports the justified assignment $_{J\vdash}L$, the same inference $J \vdash_k j_k : L$ annotated with a theory proof supports the justified assignment $_{J\vdash j_k:}L$ (see rule Deduce in Fig. 5).

Because proof terms may use identifiers of assignments, proof-carrying CDSAT manipulates assignments with identifiers, and the subset relation takes identifiers into account. In Fig. 5, the condition $J \subseteq \Gamma$ means that J is a $\mathcal{T}_k$-assignment with identifiers, and for every ${}^a(t \leftarrow c)$ in J, the pair $t \leftarrow c$ appears in Γ with identifier a. A conflict E is an assignment with identifiers such that $E \models \bot$ and $E \subseteq \Gamma$ with the same meaning. Conflict resolution rules operate on states of the form $\langle \Gamma;\ E\ ;\ c\rangle$, where c is a conflict proof term. An analysis of the rules shows that the invariants in Theorem 4.4 hold.

Theorem 4.4. *For all proof-carrying CDSAT-derivations*

- *If a trail containing $_{H\vdash j:}A$ is generated, then $H \vdash j : A$;*
- *If a conflict state $\langle \Gamma;\ E\ ;\ c\rangle$ is reached, then $E \vdash c : \bot$.*

A difference between the system in Fig. 5 and that in Fig. 1 is that in proof-carrying CDSAT the dichotomy between Fail and ConflictSolve is based on the outcome of the conflict resolution phase rather than the level of the conflict. Given Theorems 4.3 and 4.4, reaching state $\langle \Gamma;\ \emptyset\ ;\ c\rangle$ means that the input problem is unsatisfiable in $\mathcal{T}_\infty^+$ and c is a proof term witnessing its unsatisfiability. If such a state is reached, Fail fires and terminates the derivation in state $\mathsf{unsat}(c)$. It is simple to show that the conflict resolution rules in Fig. 5 reduce $\langle \Gamma;\ E\ ;\ c_1\rangle$ to a state of the form $\langle \Gamma;\ \emptyset\ ;\ c\rangle$ if and only if $\mathsf{level}_\Gamma(E) = 0$; and that they solve the conflict producing some trail Γ' different from Γ if and only if $\mathsf{level}_\Gamma(E) > 0$.

The CDSAT transition system of Fig. 1 returns unsat as soon as a conflict at level 0 is found, because it does not generate a proof. For proof-carrying CDSAT, detecting a conflict at level 0 does not suffice: in order to generate a proof of unsatisfiability of the initial assignment, proof-carrying CDSAT

Search rules				
Decide	Γ	$\longrightarrow$	$\Gamma, {}_?A$	if A is a $\mathcal{T}_k$-assignment for a term that is $\mathcal{T}_k$-*relevant* for Γ, and A is *acceptable* for the $\mathcal{T}_k$-module and its view of Γ, with $1 \le k \le n$
The next three rules share the conditions: $J \subseteq \Gamma$, $(J \vdash_k j_k : L)$, and $L \notin \Gamma$, for some k, $1 \le k \le n$.				
Deduce	Γ	$\longrightarrow$	$\Gamma, {}_{J\vdash j_k:}L$	if $\overline{L} \notin \Gamma$ and L is in $\mathcal{B}$
Fail	Γ	$\longrightarrow$	unsat(c)	if $\overline{L} \in \Gamma$ with id a, and $\langle \Gamma;\ J \cup \{{}^a\overline{L}\}\ ;\ \mathsf{cfl}(j_k, a)\rangle \Longrightarrow^* \langle \Gamma;\ \emptyset\ ;\ c\rangle$
ConflictSolve	Γ	$\longrightarrow$	Γ'	if $\overline{L} \in \Gamma$ with id a, and $\langle \Gamma;\ J \cup \{{}^a\overline{L}\}\ ;\ \mathsf{cfl}(j_k, a)\rangle \Longrightarrow^* \Gamma'$
Conflict resolution rules				
Undo				
$\langle \Gamma;\ E \uplus \{{}^aA\}\ ;\ c\rangle$		$\Longrightarrow$	$\Gamma^{\le m-1}$	if A is a first-order decision of level $m > \mathsf{level}_\Gamma(E)$
Resolve				
$\langle \Gamma;\ E \uplus \{{}^aA\}\ ;\ c\rangle$		$\Longrightarrow$	$\langle \Gamma;\ E \cup H\ ;\ \mathsf{res}(j, {}^aA.c)\rangle$	if ${}_{H\vdash j:}A$ is in Γ and H does not contain a first-order decision A' whose level is $\mathsf{level}_\Gamma(E \uplus \{A\})$
UndoDecide				
$\langle \Gamma;\ E \uplus \{{}^aL\}\ ;\ c\rangle$		$\Longrightarrow$	$\Gamma^{\le m-1}, {}_?\overline{L}$	if ${}_{H\vdash j:}L$ is in Γ, $m = \mathsf{level}_\Gamma(E) = \mathsf{level}_\Gamma(L)$ and H contains a first-order decision A' of level m
LearnBackjump				
$\langle \Gamma;\ E \uplus H\ ;\ c\rangle$		$\Longrightarrow$	$\Gamma^{\le m}, {}_{E\vdash \mathsf{lem}(H.c):}L$	if L is a clausal form of H and is in $\mathcal{B}$, $L \notin \Gamma$, $\overline{L} \notin \Gamma$, and $\mathsf{level}_\Gamma(E) \le m < \mathsf{level}_\Gamma(H)$

Figure 5. The proof-carrying CDSAT transition system

integrates a proof of why E follows from the initial assignment with a proof of why E is unsatisfiable. This is achieved by proof-carrying Resolve, which combines proof term c, witnessing the unsatisfiability of the conflict, with proof term j witnessing that one of the assignments in the conflict, named A, follows from prior assignments: A is retained in the proof term $\mathsf{res}(j, {}^aA.c)$. Proof-carrying LearnBackjump generates the proof term $\mathsf{lem}(H.c)$, recording that the learned lemma L is a clausal form of H, and turning the conflict proof term c representing a proof of unsatisfiability of $E \uplus H$ into a deduction proof term representing a proof of L from E.

4.4 Example of Proof-Carrying Derivation

We now illustrate some operations of proof-carrying CDSAT. Fig. 6 gives a refutation for the full version of the example in Fig. 2. Adding identifiers to the first conflict in Fig. 2 we get

$$ {}^{a_7}(\neg l_2 \vee \neg l_4 \vee \neg l_5),\ {}^{a_2}l_2,\ {}^{a_4}l_4,\ {}^{a_5}l_5. $$

The proof term for this conflict involves Unit Propagation:

$$ c_1 \ = \ \mathsf{cfl}(\mathsf{UP}(a_7, \{a_2, a_4\}), a_5). $$

The next step replaces l_5 in the conflict by its justification ${}^{a_6}(\neg l_4 \vee l_5)$, ${}^{a_4}l_4$, yielding (Fig. 6, line 2) the conflict proof term

$$ c_2 \ = \ \mathsf{res}(\mathsf{UP}(a_6, \{a_4\}), {}^{a_5}l_5.c_1) $$

where $\mathsf{UP}(a_6, \{a_4\})$ is the theory proof that justifies why l_5 is on the trail as a result of Unit Propagation. The conflict is solved by LearnBackjump as in Example 3.2, learning $(\neg l_2 \vee \neg l_4)$ and placing

$$ {}_{(\neg l_2 \vee \neg l_4 \vee \neg l_5),\ (\neg l_4 \vee l_5) \vdash j_1:}(\neg l_2 \vee \neg l_4) $$

on the trail (line 3), where $j_1 \ = \ \mathsf{lem}(\{{}^{a_2}l_2, {}^{a_4}l_4\}.c_2)$.

These patterns repeat several times in Fig. 6. For example, $j_2, j_5, j_6, j_9, j_{10}$ (lines 4,9,10,17,18) are deduction proof terms of the form $\mathsf{UP}(\_, \_)$, as they support unit propagations, and similarly to c_1, proof term c_8 for conflict ${}^a(\neg l_2 \vee \neg l_4)$, ${}^{a_2}l_2$, ${}^{a_4}l_4$ plugs in a unit propagation subproof (line 19):

$$ c_8 \ = \ \mathsf{cfl}(\mathsf{UP}(a, \{a_2\}), a_4). $$

The derivation involves the LRA inference (line 5)

$$ {}^{a_1}(x \leftarrow {}^3/_4) \vdash j_3 : \overline{(x \le 0)} $$

where $j_3 = \mathsf{eval}(\{a_1\})$, and the LRA inference (line 6)

$$ {}^{a_2}(y \ge 0),\ {}^{a_8}\overline{(x+y>0)},\ {}^{a_9}\overline{(x \le 0)} \vdash c_3 : \bot $$

where $c_3 = \mathsf{cfl}(\mathsf{FM}(a_2, a_8), a_9)$ as

$$ {}^{a_2}(0 \le y),\ {}^{a_8}(y \le -x) \vdash_{\mathsf{LRA}} \mathsf{FM}(a_2, a_8) : (0 \le -x). $$

It also employs the EUF inferences (lines 11 and 12)

$$ {}^{a_{10}}(f(z) \leftarrow \mathsf{blue}),\ {}^{a_{11}}(f(y) \leftarrow \mathsf{red}) \vdash j_7 : f(z) \not\simeq f(y) $$
$$ {}^{a_{12}}(z \simeq y),\ {}^{a_{13}}(f(z) \not\simeq f(y)) \vdash c_5 : \bot $$

where $j_7 = \mathsf{neq}(a_{10}, a_{11})$ and $c_5 = \mathsf{cfl}(\mathsf{Cong}(a_{12}), a_{13})$.

Deduction proof terms j_4 and j_8 (lines 8 and 16) justify the learning of lemmas $((x \le 0) \vee \neg l_2)$ and $(x \le 0)$, respectively: therefore they instantiate the same pattern as j_1, namely $\mathsf{lem}(\{\_\}.\_)$. The conflict proof terms c_4, c_6, c_7, and c_9 through c_{21} all encode Resolve steps, and therefore are of the form $\mathsf{res}(\_, \_.\_)$ like c_2. More precisely, c_{13}, $c_{15} - c_{18}$, c_{20}, and c_{21}, are of the form $\mathsf{res}(\mathsf{in}(\_), \_.\_)$, whose effect is simply to remove an assignment from the conflict because it is an initial assignment. In contrast, conflict proof term c_{19} is of the form

$$ \mathsf{res}(j_1, {}^a(\neg l_2 \vee \neg l_4).c_{18}), $$

where a is the identifier of the learned clause $(\neg l_2 \vee \neg l_4)$ (line

Input problem: $(\neg l_4 \vee l_5)$, $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$, $(l_2 \vee (z \simeq y))$, $(\neg(x \leq 0) \vee l_2)$, $(\neg(x \leq 0) \vee l_4)$, $(f(z) \leftarrow \text{blue})$, $(f(y) \leftarrow \text{red})$ where l_2 is $(y \geq 0)$, and l_4 is $(x+y > 0)$. Assume decision A_1 is $(x \leftarrow 3/4)$. Detected conflicts are introduced with symbol ⚠. To improve readability, identifiers in justifications and conflicts are omitted; they are determined by the current trail.

$\quad\quad$ ⚠ $\langle \Gamma;\ (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, l_5\ ;\ c_1 \rangle$ 1

$\quad\quad \Longrightarrow \langle \Gamma;\ (\neg l_2 \vee \neg l_4 \vee \neg l_5), l_2, l_4, (\neg l_4 \vee l_5)\ ;\ c_2 \rangle$ 2

$\Longrightarrow\ \Gamma_0,\ ?A_1,\ ?l_2,\ {\scriptstyle (\neg l_2 \vee \neg l_4 \vee \neg l_5),\ (\neg l_4 \vee l_5) \vdash j_1 :} (\neg l_2 \vee \neg l_4)$ 3

$\longrightarrow\ \Gamma_0,\ ?A_1,\ ?l_2,\ {\scriptstyle (\neg l_2 \vee \neg l_4 \vee \neg l_5),\ (\neg l_4 \vee l_5) \vdash j_1 :} (\neg l_2 \vee \neg l_4),\ {\scriptstyle (\neg l_2 \vee \neg l_4),\ l_2 \vdash j_2 :} \overline{l_4}$ 4

$\longrightarrow\ \Gamma_0,\ ?A_1,\ ?l_2,\ {\scriptstyle (\neg l_2 \vee \neg l_4 \vee \neg l_5),\ (\neg l_4 \vee l_5) \vdash j_1 :} (\neg l_2 \vee \neg l_4),\ {\scriptstyle (\neg l_2 \vee \neg l_4),\ l_2 \vdash j_2 :} \overline{l_4},\ {\scriptstyle A_1 \vdash j_3 :} \overline{(x \leq 0)} \quad =: \Gamma_1$ 5

$\quad\quad$ ⚠ $\langle \Gamma_1;\ l_2, \overline{l_4}, \overline{(x \leq 0)}\ ;\ c_3 \rangle$ 6

$\quad\quad \Longrightarrow \langle \Gamma_1;\ l_2, (\neg l_2 \vee \neg l_4), \overline{(x \leq 0)}\ ;\ c_4 \rangle$ 7

$\Longrightarrow\ \Gamma_0,\ ?A_1,\ {\scriptstyle (\neg l_2 \vee \neg l_4 \vee \neg l_5),\ (\neg l_4 \vee l_5) \vdash j_1 :} (\neg l_2 \vee \neg l_4),\ {\scriptstyle A_1 \vdash j_3 :} \overline{(x \leq 0)},\ {\scriptstyle (\neg l_2 \vee \neg l_4) \vdash j_4 :} (x \leq 0) \vee \neg l_2 \quad =: \Gamma_2$ 8

$\longrightarrow\ \Gamma_2,\ {\scriptstyle ((x \leq 0) \vee \neg l_2),\ \overline{(x \leq 0)} \vdash j_5 :} \overline{l_2}$ 9

$\longrightarrow\ \Gamma_2,\ {\scriptstyle ((x \leq 0) \vee \neg l_2),\ \overline{(x \leq 0)} \vdash j_5 :} \overline{l_2},\ {\scriptstyle (l_2 \vee (z \simeq y)),\ \overline{l_2} \vdash j_6 :} (z \simeq y)$ 10

$\longrightarrow\ \Gamma_2,\ {\scriptstyle ((x \leq 0) \vee \neg l_2),\ \overline{(x \leq 0)} \vdash j_5 :} \overline{l_2},\ {\scriptstyle (l_2 \vee (z \simeq y)),\ \overline{l_2} \vdash j_6 :} (z \simeq y),\ {\scriptstyle (f(z) \leftarrow \text{blue}),\ (f(y) \leftarrow \text{red}) \vdash j_7 :} (f(z) \not\simeq f(y)) \quad =: \Gamma_3$ 11

$\quad\quad$ ⚠ $\langle \Gamma_3;\ (f(z) \not\simeq f(y)), (z \simeq y)\ ;\ c_5 \rangle$ 12

$\quad\quad \Longrightarrow \langle \Gamma_3;\ (f(z) \not\simeq f(y)), (l_2 \vee (z \simeq y)), \overline{l_2}\ ;\ c_6 \rangle$ 13

$\quad\quad \Longrightarrow \langle \Gamma_3;\ (f(z) \not\simeq f(y)), (l_2 \vee (z \simeq y)), ((x \leq 0) \vee \neg l_2), \overline{(x \leq 0)}\ ;\ c_7 \rangle$ 14

$\Longrightarrow\ \Gamma_0,\ {\scriptstyle (\neg l_2 \vee \neg l_4 \vee \neg l_5),\ (\neg l_4 \vee l_5) \vdash j_1 :} (\neg l_2 \vee \neg l_4),\ {\scriptstyle (\neg l_2 \vee \neg l_4) \vdash j_4 :} (x \leq 0) \vee \neg l_2,\ {\scriptstyle (f(z) \leftarrow \text{blue}),\ (f(y) \leftarrow \text{red}) \vdash j_7 :} (f(z) \not\simeq f(y)),$ 15

$\quad\quad {\scriptstyle (f(z) \not\simeq f(y)),\ (l_2 \vee (z \simeq y)),\ ((x \leq 0) \vee \neg l_2) \vdash j_8 :} (x \leq 0) \quad =: \Gamma_4$ 16

$\longrightarrow\ \Gamma_4,\ {\scriptstyle (\neg(x \leq 0) \vee l_2),\ (x \leq 0) \vdash j_9 :} l_2$ 17

$\longrightarrow\ \Gamma_4,\ {\scriptstyle (\neg(x \leq 0) \vee l_2),\ (x \leq 0) \vdash j_9 :} l_2,\ {\scriptstyle (\neg(x \leq 0) \vee l_4),\ (x \leq 0) \vdash j_{10} :} l_4 \quad =: \Gamma_5$ 18

$\quad\quad$ ⚠ $\langle \Gamma_5;\ (\neg l_2 \vee \neg l_4), l_2, l_4\ ;\ c_8 \rangle$ 19

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4), l_2, (\neg(x \leq 0) \vee l_4), (x \leq 0)\ ;\ c_9 \rangle$ 20

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4), (\neg(x \leq 0) \vee l_2), (\neg(x \leq 0) \vee l_4), (x \leq 0)\ ;\ c_{10} \rangle$ 21

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4), (\neg(x \leq 0) \vee l_2), (\neg(x \leq 0) \vee l_4), (f(z) \not\simeq f(y)), (l_2 \vee (z \simeq y)), ((x \leq 0) \vee \neg l_2)\ ;\ c_{11} \rangle$ 22

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4), (\neg(x \leq 0) \vee l_2), (\neg(x \leq 0) \vee l_4), (f(z) \not\simeq f(y)), (l_2 \vee (z \simeq y))\ ;\ c_{12} \rangle$ 23

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4), (\neg(x \leq 0) \vee l_2), (\neg(x \leq 0) \vee l_4), (f(z) \not\simeq f(y))\ ;\ c_{13} \rangle$ 24

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4), (\neg(x \leq 0) \vee l_2), (\neg(x \leq 0) \vee l_4), (f(z) \leftarrow \text{blue}), (f(y) \leftarrow \text{red})\ ;\ c_{14} \rangle$ 25

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4), (\neg(x \leq 0) \vee l_2), (\neg(x \leq 0) \vee l_4), (f(z) \leftarrow \text{blue})\ ;\ c_{15} \rangle$ 26

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4), (\neg(x \leq 0) \vee l_2), (\neg(x \leq 0) \vee l_4)\ ;\ c_{16} \rangle$ 27

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4), (\neg(x \leq 0) \vee l_2)\ ;\ c_{17} \rangle$ 28

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4)\ ;\ c_{18} \rangle$ 29

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4 \vee \neg l_5), (\neg l_4 \vee l_5)\ ;\ c_{19} \rangle$ 30

$\quad\quad \Longrightarrow \langle \Gamma_5;\ (\neg l_2 \vee \neg l_4 \vee \neg l_5)\ ;\ c_{20} \rangle$ 31

$\quad\quad \Longrightarrow \langle \Gamma_5;\ \emptyset\ ;\ c_{21} \rangle$ 32

$\longrightarrow\ \text{unsat}(c_{21})$ 33

Figure 6. A proof-carrying continuation of the CDSAT derivation in Fig. 2

30). Note that a occurs twice in c_{19}: once in the subproof c_8 (line 19), as seen above, and once in the subproof (line 23)

$$c_{12} = \text{res}(j_4, {}^{a_{14}}((x \leq 0) \vee \neg l_2).c_{11}),$$

since $j_4 = \text{lem}(\{{}^{a_9}\overline{(x \leq 0)}, {}^{a_2} l_2\}.c_4)$ (line 8), $c_4 = \text{res}(j_2, {}^{a_8}\overline{l_4}.c_3)$ (line 7), and $j_2 = \text{UP}(a, \{a_2\})$ (line 4). This double occurrence reflects the fact that lemma $(\neg l_2 \vee \neg l_4)$ has been *reused* in the derivation, which is a purpose of learning. Had Backjump been applied instead of LearnBackjump (cf. Example 3.4), the trail would have been extended (line 3) with

$${\scriptstyle (\neg l_2 \vee \neg l_4 \vee \neg l_5),\ (\neg l_4 \vee l_5),\ l_2 \vdash j_{11} :} \overline{l_4}$$

where the deduction proof $j_{11} = \text{lem}(\{{}^{a_4} l_4\}.c_2)$ justifies why $\overline{l_4}$ is on the trail. If this route had been followed, the conflict reflected by c_8 (line 19) would not have been detected. More steps would have been necessary to discover it, amounting to constructing another proof of $(\neg l_2 \vee \neg l_4)$, possibly identical to the first one that was not recorded.

A full development of the final conflict proof term c_{21} would show the absence of the deduction proof term j_3

(introduced on line 5). This is a general phenomenon in CDSAT: first-order decisions play a role in finding models, not proofs; thus, deduction proof terms referring to first-order decisions cannot appear in the final proof term, and an easy optimization is to avoid constructing them.

5 From Proof Terms to Proofs

The CDSAT proof terms record enough information to reconstruct a proof of unsatisfiability from a CDSAT derivation concluding unsat. The proof system in Fig. 4 describes how proof terms specify the proof reconstruction steps to be performed. A deduction proof term j with $H \vdash j : A$ indicates how to reconstruct a proof of $H \models A$. A conflict proof term c with $E \vdash c : \bot$ indicates how to reconstruct a proof of $E \models \bot$. For instance, the rule for $\mathsf{res}(j, {}^a A.c)$ can be read as: if j allows the reconstruction of a proof that A follows from H, and if c allows the reconstruction of a proof that $E, {}^a A$ is unsatisfiable, then $\mathsf{res}(j, {}^a A.c)$ allows the reconstruction of a proof that $E \cup H$ is unsatisfiable. A solver implementing the proof-carrying CDSAT transition system can reconstruct the proof in any proof format that (1) features two kinds of proofs (deduction proofs and conflict proofs), and (2) is equipped with proof operations corresponding to the rules of Fig. 4. In this view, proof reconstruction *interprets* proof terms in the desired proof format.

An alternative option is that the CDSAT solver manipulates directly the proofs denoted by the expressions $\mathsf{in}(A)$, j_k, $\mathsf{lem}(H.c)$, $\mathsf{cfl}(j_k, a)$, and $\mathsf{res}(j, {}^a A.c)$. The choice between the two approaches depends on whether the proofs generated by proof reconstruction are better (e.g., shorter, more readable) than the proof terms. Another factor is whether direct manipulation during the derivation causes such a high runtime overhead that proof reconstruction is preferable. A third possibility is to consider proof terms themselves as the proof objects to be produced. Then a proof-checker would check directly the proof terms, checking that the rules of Fig. 4 are applied correctly. In the the rest of this section, we see how CDSAT proof terms can be turned into resolution-style proofs (Subsection 5.1), and how the LCF approach to proofs can be applied to CDSAT (Subsection 5.2).

5.1 Proof Format Based on Resolution

Resolution proofs are based on clausal form. A few caveats are in order. First, CDSAT views logical connectives such as $\wedge$ and $\vee$ as interpreted symbols of the Boolean theory. Second, the existence of assignments such as $(l_1 \vee l_2) \leftarrow \mathsf{false}$ implies that clauses $l_1 \vee \cdots \vee l_m$ may not be the only formulæ in proofs. Third, CDSAT does *not* assume that its input is in Conjunctive Normal Form (CNF). Therefore, we distinguish between *object-level clauses*, that are the clauses $l_1 \vee \cdots \vee l_m$, and *CDSAT clauses*, that are the clauses in *resolution proofs reconstructed from CDSAT proof terms*. A *CDSAT clause*, or *clause* for short when there is no ambiguity, is a set of singleton Boolean assignments, written $L_1 \,||\, \cdots \,||\, L_m$ to avoid confusion with object-level clauses. A CDSAT clause is a clause in the standard propositional sense, with CDSAT formulæ (i.e., terms of sort prop) in place of Boolean variables (aka propositional atoms). Since in CDSAT there are first-order assignments, the resolution proofs reconstructed from CDSAT proof terms use *guarded clauses*, written $H \rightarrow C$, where H is a set of first-order assignments and C is a CDSAT clause. Guarded or hypothetical clauses are used extensively in the literature (e.g., [9, 15, 16]).

A *resolution proof reconstructed from a CDSAT proof term* is a binary tree such that:

- Every leaf is labeled with either a guarded clause, called in this case *theory lemma*, or an input singleton assignment; and
- Every internal node is labeled with a guarded clause, and, together with its two children, forms an instance of one of the following rules:

$$\frac{H_1 \rightarrow C_1 \,||\, L \qquad H_2 \rightarrow C_1 \,||\, \overline{L}}{H_1 \cup H_2 \rightarrow C_1 \,||\, C_2}$$

$$\frac{L \qquad H \rightarrow C \,||\, \overline{L}}{H \rightarrow C} \qquad \frac{A \qquad H, A \rightarrow C}{H \rightarrow C}$$

The top rule is binary resolution, generalized to the presence of first-order assignments which play no role in the rule. In the bottom two rules, the left premise is a leaf labeled by an input singleton assignment: the left-hand rule can be seen as unit resolution; the right-hand rule covers the analogous case for a first-order assignment A.

In order to illustrate theory lemmas, consider a clausal form L_0 of $\{L_1, \ldots, L_m\}$. The Boolean theory has the following lemmas, that can label the leaves of resolution trees:

$$\frac{}{\emptyset \rightarrow \overline{L_0} \,||\, \overline{L_1} \,||\, \cdots \,||\, \overline{L_m}} \qquad \frac{}{\emptyset \rightarrow L_0 \,||\, L_i}\ 1 \le i \le m \tag{2}$$

Since L_0 is a clausal form of $\{L_1, \ldots, L_m\}$, $\overline{L_0} \,||\, \overline{L_1} \,||\, \cdots \,||\, \overline{L_m}$ and $L_0 \,||\, L_i$, $1 \le i \le m$, are propositional tautologies. The first lemma allows the transformation of an object-level clause, the assignment L_0, into a CDSAT clause:

$$\frac{\overset{\cdots}{H \rightarrow C \,||\, L_0} \qquad \emptyset \rightarrow \overline{L_0} \,||\, \overline{L_1} \,||\, \cdots \,||\, \overline{L_m}}{H \rightarrow C \,||\, \overline{L_1} \,||\, \cdots \,||\, \overline{L_m}}$$

Conversely, the other lemmas allow the reification of a CDSAT clause $\overline{L_1} \,||\, \cdots \,||\, \overline{L_m}$ into an object-level clause:

$$\frac{\dfrac{\overset{\cdots}{H \rightarrow C \,||\, \overline{L_1} \,||\, \cdots \,||\, \overline{L_m}} \qquad \emptyset \rightarrow L_0 \,||\, L_1}{\cdots} \qquad \emptyset \rightarrow L_0 \,||\, L_m}{H \rightarrow C \,||\, L_0}$$

These transformations can be expressed compactly as the derivable inference rules

$$[\![\emptyset \vdash \mathsf{in}(A) : A]\!] := \frac{}{\overline{A}}$$

$$[\![J \vdash j_k : L]\!] := \frac{}{\overline{J_{\mathsf{FO}} \rightarrow J_{\mathsf{clause}} \,||\, L}}$$

$$[\![J \uplus \{{}^a\overline{L}\} \vdash \mathsf{cfl}(j_k, a) : \bot]\!] := \frac{}{\overline{J_{\mathsf{FO}} \rightarrow J_{\mathsf{clause}} \,||\, L}}$$

$$[\![E \vdash \mathsf{res}(\mathsf{in}(L), {}^aL.c) : \bot]\!] := \frac{\overline{L} \qquad \dfrac{[\![E \uplus \{{}^aL\} \vdash c : \bot]\!]}{E_{\mathsf{FO}} \rightarrow E_{\mathsf{clause}} \,||\, \overline{L}}}{E_{\mathsf{FO}} \rightarrow E_{\mathsf{clause}}}$$

$$[\![E \vdash \mathsf{lem}(H.c) : L]\!] := \frac{\dfrac{[\![E \uplus H \vdash c : \bot]\!]}{E_{\mathsf{FO}} \rightarrow E_{\mathsf{clause}} \,||\, H_{\mathsf{clause}}}}{E_{\mathsf{FO}} \rightarrow E_{\mathsf{clause}} \,||\, L}$$

$$[\![E \cup H \vdash \mathsf{res}(j, {}^aL.c) : \bot]\!] \text{ if } j \text{ is not of the form } \mathsf{in}(A) := \frac{[\![H \vdash j : L]\!] \qquad \dfrac{[\![E \uplus \{{}^aL\} \vdash c : \bot]\!]}{E_{\mathsf{FO}} \rightarrow E_{\mathsf{clause}} \,||\, \overline{L}}}{H_{\mathsf{FO}} \cup E_{\mathsf{FO}} \rightarrow H_{\mathsf{clause}} \,||\, E_{\mathsf{clause}}}$$

$$[\![E \vdash \mathsf{res}(A, {}^aA.c) : \bot]\!] \text{ if } A \text{ is first-order} := \frac{\overline{A} \qquad \dfrac{[\![E \uplus \{{}^aA\} \vdash c : \bot]\!]}{E_{\mathsf{FO}}, A \rightarrow E_{\mathsf{clause}}}}{E_{\mathsf{FO}} \rightarrow E_{\mathsf{clause}}}$$

Figure 7. Interpretation of proof terms in the resolution proof format

$$\frac{H \rightarrow C \,||\, L_0}{H \rightarrow C \,||\, \overline{L_1} \,||\, \cdots \,||\, \overline{L_m}} \qquad \frac{H \rightarrow C \,||\, \overline{L_1} \,||\, \cdots \,||\, \overline{L_m}}{H \rightarrow C \,||\, L_0}$$

that involve the Boolean theory lemmas only if $m \geq 2$ (i.e., the clause is not a unit clause).

Given an assignment H, we write H_{FO} for the greatest subset of H made of first-order assignments; and we write H_{clause} for the clause $\overline{L_1} \,||\, \cdots \,||\, \overline{L_n}$ where $L_1, \ldots, L_n$ are the singleton Boolean assignments in H.

The resolution proof format for CDSAT interprets

- A deduction proof term $\emptyset \vdash \mathsf{in}(A) : A$ as a proof concluding A;
- A deduction proof term $H \vdash j : L$ of another form as a proof concluding $H_{\mathsf{FO}} \rightarrow H_{\mathsf{clause}} \,||\, L$;
- A conflict proof term $H \vdash c : \bot$ as a proof concluding $H_{\mathsf{FO}} \rightarrow H_{\mathsf{clause}}$.

The interpretation is defined by induction on proof terms as shown in Fig. 7. For instance, consider the interpretations of a deduction proof term and a conflict proof term that encapsulate a unit propagation, where K is a clausal form of $H = H' \uplus \{L\}$ as in inference (1) from Section 3:

$$\frac{[\![\{{}^aK\} \uplus H' \vdash \mathsf{UP}(a, \{a_1, \ldots, a_n\}) : \overline{L}]\!]}{[\![\{{}^aK\} \uplus H', {}^{a_0}L \vdash \mathsf{cfl}(\mathsf{UP}(a, \{a_1, \ldots, a_n\}), a_0) : \bot]\!]}$$

where $a_1, \ldots, a_n$ are the identifiers of the elements of H'. Both become an instance of a Boolean theory lemma (2):

$$\frac{}{\overline{\emptyset \rightarrow \overline{K} \,||\, H'_{\mathsf{clause}} \,||\, \overline{L}}}$$

Since a CDSAT answer of the form $\mathsf{unsat}(c)$ means $\emptyset \vdash c : \bot$, the resolution proof reconstructed from proof term c is a refutation, as its conclusion is $\emptyset \rightarrow \emptyset$, that is, the empty clause. As already noted in Section 4, first-order decisions do not appear in proofs. It follows that first-order assignments may appear in proofs only if they are initial assignments. If the input problem contains no first-order assignments (i.e., it is an SMT problem), the reconstructed proof involves only singleton Boolean assignments labeling leaves and guarded clauses of the form $\emptyset \rightarrow C$ labeling leaves or internal nodes. In other words, the reconstructed proof is a resolution refutation in the standard sense, deriving the empty clause, and with leaves labeled by input assignments or theory lemmas.

The use of theory lemmas for propositional reasoning is less usual and it descends from the choice of treating propositional logic as a theory. Advantages include the possibility of applying CDSAT to problems that are not in CNF, and the possibility of transforming object-level clauses into CDSAT clauses and vice versa, as shown above. These transformations also provide a native feature for *sharing* resolution proofs. For instance in the refutation of Section 4.4, the conflict proof term $c_{19} = \mathsf{res}(j_1, {}^a(\neg l_2 \vee \neg l_4).c_{18})$, with $j_1 = \mathsf{lem}(\{{}^{a_2}l_2, {}^{a_4}l_4\}.c_2)$, yields ${}^{a_6}L_1, {}^{a_7}L_2 \vdash c_{19} : \bot$, where L_1 is $(\neg l_4 \vee l_5)$ and L_2 is $(\neg l_2 \vee \neg l_4 \vee \neg l_5)$. The resolution proof reconstructed from c_{19} is:

$$\frac{\dfrac{\dfrac{[\![{}^{a_6}L_1, {}^{a_7}L_2, {}^{a_2}l_2, {}^{a_4}l_4 \vdash c_2 : \bot]\!]}{\emptyset \rightarrow \overline{L_1} \,||\, \overline{L_2} \,||\, \overline{l_2} \,||\, \overline{l_4}}}{\emptyset \rightarrow \overline{L_1} \,||\, \overline{L_2} \,||\, (\neg l_2 \vee \neg l_4)} \qquad \dfrac{[\![{}^a(\neg l_2 \vee \neg l_4) \vdash c_{18} : \bot]\!]}{\emptyset \rightarrow \overline{(\neg l_2 \vee \neg l_4)}}}{\emptyset \rightarrow \overline{L_1} \,||\, \overline{L_2}}$$

The fact that the learned clause $\neg l_2 \vee \neg l_4$ is reused in c_{18}, as described in Section 4.4, means that the resolution proof $[\![{}^a(\neg l_2 \vee \neg l_4) \vdash c_{18} : \bot]\!]$ has two leaves labeled by the same theory lemma

$$\emptyset \rightarrow \overline{(\neg l_2 \vee \neg l_4)} \,||\, \overline{l_2} \,||\, \overline{l_4}.$$

An alternative resolution proof with root $\emptyset \rightarrow \overline{L_1} \,||\, \overline{L_2}$ can be obtained by replacing those two leaves with the subproof interpreting c_2, and replacing $\overline{(\neg l_2 \vee \neg l_4)}$ by $\overline{L_1} \,||\, \overline{L_2}$ in all nodes underneath. This avoids the explicit conversions between the object-level clause $\neg l_2 \vee \neg l_4$ and the CDSAT clause $\overline{l_2} \,||\, \overline{l_4}$. However, in this alternative proof, the subtree interpreting c_2 is duplicated. Such duplications happen in the standard format of resolution trees where there is only one kind of clauses. By distinguishing between object-level clauses and CDSAT clauses, and using the former for input clauses and

the latter for generated clauses, CDSAT natively supports the explicit sharing of subproofs as in resolution DAG's.

5.2 An LCF Architecture for CDSAT

Another example of proof format is the "dummy" one, where proofs do not contain any information other than *what they are supposed to be the proofs of*:

1. A deduction proof j with $H \vdash j : L$ is the pair $\langle H, L\rangle$, and
2. A conflict proof c with $H \vdash c : \bot$ is H.

Although this "proof format" does not allow any proof checking, the trustworthiness of a reasoner producing such proofs can still be established by the LCF programming abstraction [17, 30]. This approach uses a type `theorem` , whose constructed inhabitants are provable formulæ. Actually, this type is defined as an alias for the type `formula` of formulæ, but this is known only to a fixed and well-identified piece of code, called the *LCF kernel*. This kernel hides the definition of `theorem` to the outside world and exports a range of kernel primitives to manipulate inhabitants of type `theorem` in a safe and provably correct way. For instance, a primitive

```
modus_ponens : theorem -> theorem -> theorem
```

takes as arguments two inhabitants F and G of type `theorem`, checks that F is of the form $G \Rightarrow R$, and returns R as an inhabitant of `theorem`. The kernel can export a primitive that reveals that any inhabitant of `theorem` is a formula, but obviously not one that casts any inhabitant of `formula` into type `theorem`. In this way, the existence of an inhabitant F of `theorem` witnesses the fact that F is provable, as an inhabitant only results from a series of correct manipulations by the kernel primitives: if the kernel code is trusted, then F can be trusted to be a theorem, while no proof has ever been constructed in memory.

CDSAT is well-suited for the LCF approach: given a type `assign` for assignments and `single_assign` for singleton assignments, a trusted kernel defines types

```
type deduction = assign*single_assign
type conflict = assign
```

hides their definitions to the outside world, and exports a range of primitives corresponding to the proof term constructs. The signature in Fig. 8 lists hidden type definitions and exported primitives. The primitives check that the conditions of the rules in Fig. 4 are met: for instance, `in` checks that its argument is one of the initial assignments. Primitive `lem` takes as arguments a conflict E and an assignment H, checks that H is Boolean and is included in E, computes a clausal form L of H, and produces the deduction $\langle E \backslash H, L\rangle$, where $\backslash$ is set subtraction. Primitive `res` takes a deduction $\langle H, L\rangle$ and a conflict, checks that L appears in the conflict, and returns the conflict where L is replaced by H. Primitives `coerc` and `cfl` take as arguments a $\mathcal{T}_k$-theory proof, $1 \leq k \leq n$, given as an inhabitant of `'k theory_proof`, a type parameterized by k. Their first argument is a handler for theory $\mathcal{T}_k$, whose type `'k theory_handler` is parameterized by

```
type deduction
type conflict
in : single_assign -> deduction
coerc : 'k theory_handler
        -> 'k theory_proof -> deduction
lem : conflict -> assign -> deduction
cfl : 'k theory_handler
          -> 'k theory_proof -> conflict
res : deduction -> conflict -> conflict
reveal : conflict -> assign
```

Figure 8. API exported by a CDSAT kernel

a matching k, as implemented for example by a Generalized Algebraic DataType (GADT) [11, 37]. The handler allows the primitives to check that $\mathcal{T}_k$ is one of the combined theories before coercing the theory proof, trusted to be correct, into a deduction or a conflict. Proof-carrying CDSAT can be programmed on top of this kernel, so that, when it halts with answer unsat(c), the proof term c is an inhabitant of type `conflict`. The `reveal` primitive applied to c will return the empty assignment. Although no proof has been constructed in memory, the answer is *correct by construction*.

6 Discussion

Conflict-driven satisfiability procedures work by constructing partial assignments, detecting conflicts when the assignment falsifies the input formula, and performing conflict-driven inferences to learn new formulæ and reorient the search. In prior work, we presented CDSAT as a combination procedure for conflict-driven theory satisfiability solvers for mutually disjoint theories, and proved its termination, soundness, and completeness under suitable assumptions. In the present work, we extend the CDSAT transition system with lemmatization so that new learned clauses can be added during backjumping from a conflict even in the context of a partial assignment. We also annotate the transition system in order to construct a proof object when the procedure discovers that the input problem is unsatisfiable. These proof objects can be rendered in a number of proof formats and the resulting proofs checked by a trusted checker or shown to be correct by construction.

In future work, we plan to implement a reasoner based on the CDSAT approach exploring different search plans and proof formats. Search plans play a key role in coordinating the work of theory modules, including prioritizing them with respect to both decisions and deductions. Topics for further investigations include extensions of proofs to account for preprocessing techniques widely used in SAT and SMT solving, evaluating the cost of proof generation and proof checking in CDSAT, and studying proof formats that reduce it as done for instance in SAT solving [13].

References

[1] Michael Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. 2011. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In *Proceedings of the First International Conference on Certified Programs and Proofs (CPP)*, Jean-Pierre Jouannaud and Zhong Shao (Eds.). Springer, Heidelberg, Germany, EU, 135–150.

[2] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 2006. Splitting on demand in SAT modulo theories. In *Proceedings of the Thirteenth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) (Lecture Notes in Artificial Intelligence)*, Miki Hermann and Andrei Voronkov (Eds.), Vol. 4246. Springer, Heidelberg, Germany, EU, 512–526.

[3] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. 2011. Extending Sledgehammer with SMT solvers. In *Proceedings of the Twenty-Third International Conference on Automated Deduction (CADE) (Lecture Notes in Artificial Intelligence)*, Nikolaj Bjørner and Viorica Sofronie-Stokkermans (Eds.), Vol. 6803. Springer, Heidelberg, Germany, EU, 116–130.

[4] Sascha Böhme and Tjark Weber. 2010. Fast LCF-style proof reconstruction for Z3. In *Proceedings of the International Conference on Interactive Theorem Proving (ITP) (Lecture Notes in Computer Science)*, Matt Kaufmann and Lawrence C. Paulson (Eds.), Vol. 6172. Springer, Heidelberg, Germany, EU, 179–194.

[5] Maria Paola Bonacina. 1996. On the reconstruction of proofs in distributed theorem proving: a modified Clause-Diffusion method. *Journal of Symbolic Computation* 21, 4–6 (1996), 507–522.

[6] Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar. 2016. *A model-constructing framework for theory combination.* Technical Report 99/2016. Dipartimento di Informatica, Università degli Studi di Verona, Verona, Italy, EU. https://hal.archives-ouvertes.fr/hal-01425305 Also Technical Report of SRI International and INRIA - CNRS - École Polytechnique; revised November 2017.

[7] Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar. 2017. Satisfiability modulo theories and assignments. In *Proceedings of the Twenty-Sixth International Conference on Automated Deduction (CADE) (Lecture Notes in Artificial Intelligence)*, Leonardo de Moura (Ed.), Vol. 10395. Springer, Heidelberg, Germany, EU, 42–59.

[8] Maria Paola Bonacina and Moa Johansson. 2015. Interpolation systems for ground proofs in automated deduction: a survey. *Journal of Automated Reasoning* 54, 4 (2015), 353–390.

[9] Maria Paola Bonacina, Christopher A. Lynch, and Leonardo de Moura. 2011. On deciding satisfiability by theorem proving with speculative inferences. *Journal of Automated Reasoning* 47, 2 (2011), 161–189.

[10] Ritu Chadha and David A. Plaisted. 1993. On the mechanical derivation of loop invariants. *Journal of Symbolic Computation* 15, 5-6 (1993), 705–744.

[11] James Cheney and Ralf Hinze. 2003. *First-class phantom types.* Technical Report CUCIS TR2003-1901. Cornell University, Ithaca, NY, USA.

[12] Scott Cotton. 2010. Natural domain SMT: A preliminary assessment. In *Proceedings of the Eighth International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS) (Lecture Notes in Computer Science)*, Krishnendu Chatterjee and Thomas A. Henzinger (Eds.), Vol. 6246. Springer, Heidelberg, Germany, EU, 77–91.

[13] Luís Cruz-Felipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. 2017. Efficient certified RAT verification. In *Proceedings of the Twenty-Sixth International Conference on Automated Deduction (CADE) (Lecture Notes in Artificial Intelligence)*, Leonardo de Moura (Ed.), Vol. 10395. Springer, Heidelberg, Germany, EU, 220–236.

[14] Leonardo de Moura and Dejan Jovanović. 2013. A model-constructing satisfiability calculus. In *Proceedings of the Fourteenth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI) (Lecture Notes in Computer Science)*, Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni (Eds.), Vol. 7737. Springer, Heidelberg, Germany, EU, 1–12.

[15] Michael Dierkes. 2001. *Model Building for Sets of Guarded Clauses.* Ph.D. Dissertation. Institut National Polytechnique de Grenoble, Grenoble, France, EU.

[16] Harald Ganzinger and Hans de Nivelle. 1999. A superposition decision procedure for the guarded fragment with equality. In *Proceedings of the Fourteenth IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society Press, Los Alamitos, CA, USA.

[17] Michael Gordon, Robin Milner, and Christopher Wadsworth. 1979. *Edinburgh LCF: a mechanized logic of computation.* Lecture Notes in Computer Science, Vol. 78. Springer, Heidelberg, Germany, EU.

[18] Stéphane Graham-Lengrand. 2013. Psyche: a proof-search engine based on sequent calculus with an LCF-style architecture. In *Proceedings of the Twenty-Second International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX) (Lecture Notes in Artificial Intelligence)*, Didier Galmiche and Dominique Larchey-Wendling (Eds.), Vol. 8123. Springer, Heidelberg, Germany, EU, 149–156.

[19] Leopold Haller, Alberto Griggio, Martin Brain, and Daniel Kroening. 2012. Deciding floating-point logic with systematic abstraction. In *Proceedings of the Twelfth International Conference on Formal Methods in Computer Aided Design (FMCAD)*, Gianpiero Cabodi and Satnam Singh (Eds.). ACM and IEEE, New York, NY, USA.

[20] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. 2004. Abstractions from proofs. In *Proceedings of the Thirty-First ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, Xavier Leroy (Ed.). ACM, New York, NY, USA, 232–244.

[21] Dejan Jovanović. 2017. Solving nonlinear integer arithmetic with MCSAT. In *Proceedings of the Eighteenth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI) (Lecture Notes in Computer Science)*, Ahmed Bouajjani and David Monniaux (Eds.), Vol. 10145. Springer, Heidelberg, Germany, EU, 330–346.

[22] Dejan Jovanović, Clark Barrett, and Leonardo de Moura. 2013. The design and implementation of the model-constructing satisfiability calculus. In *Proceedings of the Thirteenth Conference on Formal Methods in Computer Aided Design (FMCAD)*, Barbara Jobstman and Sandip Ray (Eds.). ACM and IEEE, New York, NY, USA.

[23] Dejan Jovanović and Leonardo de Moura. 2011. Cutting to the chase: solving linear integer arithmetic. In *Proceedings of the Twenty-Third International Conference on Automated Deduction (CADE) (Lecture Notes in Artificial Intelligence)*, Nikolaj Bjørner and Viorica Sofronie-Stokkermans (Eds.), Vol. 6803. Springer, Heidelberg, Germany, EU, 338–353.

[24] Dejan Jovanović and Leonardo de Moura. 2012. Solving non-linear arithmetic. In *Proceedings of the Sixth International Joint Conference on Automated Reasoning (IJCAR) (Lecture Notes in Artificial Intelligence)*, Bernhard Gramlich, Dale Miller, and Ulrike Sattler (Eds.), Vol. 7364. Springer, Heidelberg, Germany, EU, 339–354.

[25] Konstantin Korovin, Nestan Tsiskaridze, and Andrei Voronkov. 2009. Conflict resolution. In *Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP) (Lecture Notes in Computer Science)*, Ian P. Gent (Ed.), Vol. 5732. Springer, Heidelberg, Germany, EU, 509–523.

[26] Sava Krstić and Amit Goel. 2007. Architecting solvers for SAT modulo theories: Nelson-Oppen with DPLL. In *Proceedings of the Sixth International Symposium on Frontiers of Combining Systems (FroCoS) (Lecture Notes in Artificial Intelligence)*, Frank Wolter (Ed.), Vol. 4720. Springer, Heidelberg, Germany, EU, 1–27.

[27] João P. Marques Silva, Inês Lynce, and Sharad Malik. 2009. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*, Armin Biere, Marjin Heule, Hans Van Maaren, and Toby Walsh (Eds.). Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press, Amsterdam, The Netherlands, EU, 131–153.

[28] João P. Marques Silva and Karem A. Sakallah. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Comput.* 48, 5 (1999), 506–521.

[29] Kenneth L. McMillan, A. Kuehlmann, and Mooly Sagiv. 2009. Generalizing DPLL to richer logics. In *Proceedings of the Twenty-First International Conference on Computer Aided Verification (CAV) (Lecture Notes in Computer Science)*, Ahmed Bouajjani and Oded Maler (Eds.), Vol. 5643. Springer, Heidelberg, Germany, EU, 462–476.

[30] Robin Milner. 1979. LCF: A Way of Doing Proofs with a Machine. In *Proceedings of the Eighth International Symposium on Mathematical Foundations of Computer Science (MFCS) (Lecture Notes in Computer Science)*, Jirí Becvár (Ed.), Vol. 74. Springer, Heidelberg, Germany, EU, 146–159.

[31] Greg Nelson and Derek C. Oppen. 1979. Simplification by Cooperating Decision Procedures. *ACM Transactions on Programming Languages and Systems* 1, 2 (1979), 245–257.

[32] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 2006. 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.

[33] Natarajan Shankar. 2005. Inference Systems for Logical Algorithms. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science (Lecture Notes in Computer Science)*, R. Ramanujam and Sandeep Sen (Eds.), Vol. 3821. Springer, Heidelberg, Germany, EU, 60–78.

[34] Natarajan Shankar. 2008. Trust and automation in verification tools. In *Sixth International Symposium on Automated Technology for Verification and Analysis (ATVA) (Lecture Notes in Computer Science)*, Sungdeok (Steve) Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan (Eds.), Vol. 5311. Springer, Heidelberg, Germany, EU, 4–17.

[35] Chao Wang, Franjo Ivančić, Malay Ganai, and Aarti Gupta. 2005. Deciding separation logic formulae by SAT and incremental negative cycle elimination. In *Proceedings of the Twelfth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) (Lecture Notes in Artificial Intelligence)*, Geoff Sutcliffe and Andrei Voronkov (Eds.), Vol. 3835. Springer, Heidelberg, Germany, EU, 322–336.

[36] Steven A. Wolfman and Daniel S. Weld. 1999. The LPSAT engine and its application to resource planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Thomas Dean (Ed.), Vol. 1. Morgan Kaufmann Publishers, San Francisco, CA, USA, 310–316.

[37] Hongwei Xi, Chiyan Chen, and Gang Chen. 2003. Guarded recursive datatype constructors. In *Proceedings of the Thirtieth SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, Alex Aiken and Greg Morrisett (Eds.). ACM, New York, NY, USA, 224–235.

[38] Aleksandar Zeljić, Christoph M. Wintersteiger, and Philipp Rümmer. 2016. Deciding bit-vector formulas with `mcSAT`. In *Proceedings of the Nineteenth International Conference on Theory and Applications of Satisfiability Testing (SAT) (Lecture Notes in Computer Science)*, Nadia Creignou and Daniel Le Berre (Eds.), Vol. 9710. Springer, Heidelberg, Germany, EU, 249–266.

[39] Lintao Zhang and Sharad Malik. 2003. Validating SAT solvers using an independent resolution-based checker: practical implementations and other applications. In *Proceedings of the Conference on Design Automation and Test in Europe (DATE)*. IEEE, Los Alamitos, CA, USA, 10880–10885.