# A Model-Constructing Satisfiability Calculus

Leonardo de Moura[1] and Dejan Jovanović[2]

[1] Microsoft Research
[2] New York University

**Abstract.** We present a new calculus where recent model-based decision procedures and techniques can be justified and combined with the standard DPLL(T) approach to satisfiability modulo theories. The new calculus generalizes the ideas found in CDCL-style propositional SAT solvers to the first-order setting.

## 1 Introduction

Considering the theoretical hardness of SAT, the astonishing adeptness of SAT solvers when attacking practical problems has changed the way we perceive the limits of algorithmic reasoning. Modern SAT solvers are based on the idea of *conflict driven clause learning* (CDCL) [11,15,13]. The CDCL algorithm is a combination of an explicit backtracking search for a satisfying assignment complemented with a deduction system based on Boolean resolution. In this combination, the worst-case complexity of both components is circumvented by the components guiding and focusing each other. The generalization of the SAT problem into the first-order domain is called satisfiability modulo theories (SMT). The common way to solve an SMT problem is to employ a SAT solver to enumerate the assignment of the Boolean abstraction of the formula. The candidate Boolean assignment is then either confirmed or refuted by a *decision procedure* dedicated to reasoning about conjunctions of theory-specific constraints. This framework is commonly called DPLL(T) [10,14] and is employed by most of the SMT solvers today. Although DPLL(T) at its core relies on a CDCL SAT solver, this SAT solver is only used as a black-box. This can be seen as an advantage since the advances in SAT easily transfer to performance improvements in SMT. On the other hand, in the last few years the idea of direct model construction complemented with conflict resolution has been successfully generalized to fragments of SMT dealing with theories such as linear real arithmetic [4,12,9], linear integer arithmetic [7], nonlinear arithmetic [8], and floating-point [6]. All these procedures, although quite effective in their corresponding first-order domains, have not seen a more widespread acceptance due to their limitations in purely Boolean reasoning and incompatibility with DPLL(T).

In this paper we propose a *model-constructing satisfiability calculus* (mcSAT) that encompasses all the decision procedures above, including the decision procedures aimed at DPLL(T), while resolving the limitations mentioned above. The mcSAT framework extends DPLL(T) by allowing assignments of variables to concrete values, while relaxing the restriction that decisions, propagations, and explanations of conflicts must be in term of existing atoms.

## 2 A Model Based Abstract Procedure

We assume that the reader is familiar with the usual notions and terminology of first-order logic and model theory (for an introduction see e.g. [2]). We describe the new procedure as an abstract transition system in the spirit of Abstract DPLL [14]. The crucial difference of the system we present is that we are not restricted to Boolean decisions. Instead, we allow the model that the theory is trying to construct to be involved in the search and in explaining the conflicts, while allowing new literals to be introduced so as to support more complex conflict analyses.

The states in the transition system are pairs of the form $\langle M, \mathcal{C} \rangle$, where $M$ is a sequence (usually called a *trail*) of trail elements, and $\mathcal{C}$ is a set of clauses. Each trail element is either a *decided literal*, a *propagated literal*, or a *model assignment*. We refer to both decided literals and model assignments as *decisions*. A decided literal is a literal that we assume to be true. On the other hand, a propagated literal, denoted as $C{\to}L$, marks a literal $L$ that is implied to be true in the current state by the clause $C$ (the explanation). In both cases, we say that the literal $L$ appears in $M$, and write this as $L \in M$. A model assignment, written as $x \mapsto \alpha$, is an assignment of a first-order uninterpreted symbol $x$ to a value $\alpha$.[3] Given a trail $M$ that contains model assignments $x_{i_1} \mapsto \alpha_1, \ldots, x_{i_k} \mapsto \alpha_k$, we can construct a first-order interpretation $\upsilon[M] = [x_{i_1} \mapsto \alpha_1, \ldots, x_{i_k} \mapsto \alpha_k]$. Given a term $t$, the interpretation $\upsilon[M](t)$ is either a value of the term $t$ under the assignment in $M$, or $\mathsf{undef}$ if the the term cannot be fully evaluated.

The content of the trail implies an interpretation of literals and is the core of our procedure. In order to evaluate the value of some literal $L$ with respect to a trail $M$, we define the functions $\mathsf{value}_{\mathrm{B}}$ and $\mathsf{value}_{\mathrm{T}}$, the former interpreting the literal according to the Boolean assignment, and the latter interpreting the literal according to the model assignment of variables.

$$\mathsf{value}_{\mathrm{B}}(L, M) = \begin{cases} \mathsf{true} & L \in M \\ \mathsf{false} & \neg L \in M \\ \mathsf{undef} & \text{otherwise} \end{cases} \qquad \mathsf{value}_{\mathrm{T}}(L, M) = \begin{cases} \mathsf{true} & \upsilon[M](L) = \mathsf{true} \\ \mathsf{false} & \upsilon[M](L) = \mathsf{false} \\ \mathsf{undef} & \text{otherwise} \end{cases}$$

We say that a trail $M$ is *consistent* if the Boolean assignment and first-order model are not in conflict, i.e. when for all $L \in M$ we have that $\mathsf{value}_{\mathrm{T}}(L, M) \neq \mathsf{false}$. Additionally we say that the trail $M$ is *complete* when each asserted first-order literal $L \in M$ is justified by the first-order interpretation, i.e. $\mathsf{value}_{\mathrm{T}}(L, M) = \mathsf{true}$. We use the predicate $\mathsf{consistent}(M)$ to denote that $M$ is consistent and $\mathsf{complete}(M)$ to denote that $M$ is complete. Note that if a trail $M$ is consistent, this does not mean that the assertions on the trail are truly satisfiable (feasible), just that the current partial assignment does not refute any of the individual trail literals. When there is a set of literals on the trail $M$ that, together with

[3] The actual representation for values is theory specific and depends on the type of $x$. For example, for the theory of liner real arithmetic, the values are rational numbers. We never assign Boolean variables to values as they are considered literals.

the model assignments from $M$, is not satisfiable, we call the trail *infeasible* and denote this with the predicate $\mathsf{infeasible}(M)$. We have that $\neg\,\mathsf{consistent}(M)$ implies $\mathsf{infeasible}(M)$.

Since the values of $\mathsf{value}_{\mathrm{T}}(L, M)$ and $\mathsf{value}_{\mathrm{B}}(L, M)$ do not disagree for all $L \in M$, we define the value of a literal in a consistent state as

$$\mathsf{value}(L, M) = \begin{cases} \mathsf{value}_{\mathrm{B}}(L, M) & \mathsf{value}_{\mathrm{B}}(L, M) \neq \mathsf{undef}, \\ \mathsf{value}_{\mathrm{T}}(L, M) & \text{otherwise.} \end{cases}$$

*Example 1.* Consider the trail $M = [\![x > 0,\ x \mapsto 1,\ y \mapsto 0,\ z > 0]\!]$. The model interpretation according to $M$ is $v[M] = [x \mapsto 1,\ y \mapsto 0]$. Therefore we have that $\mathsf{value}_{\mathrm{T}}(x > 0, M) = \mathsf{value}_{\mathrm{B}}(x > 0, M) = \mathsf{true}$, $\mathsf{value}_{\mathrm{T}}(x > 1, M) = \mathsf{false}$, $\mathsf{value}_{\mathrm{T}}(z > 0, M) = \mathsf{undef}$, $\mathsf{value}_{\mathrm{B}}(z > 0, M) = \mathsf{true}$, $v[M](x + y + 1) = 2$, and $v[M](x + z) = \mathsf{undef}$. The trail $M$ is consistent, but $M' = [\![M, y < 0]\!]$ is not because $\mathsf{value}_{\mathrm{T}}(y < 0, M') = \mathsf{false}$ and $\mathsf{value}_{\mathrm{B}}(y < 0, M') = \mathsf{true}$. The trail $M$ is not complete as it does not interpret $z$ and therefore $\mathsf{value}_{\mathrm{T}}(z > 0, M) = \mathsf{undef}$. Finally, $M'' = [\![M, z < x]\!]$ is infeasible because $\{x \mapsto 1,\ z > 0,\ z < x\}$ is unsatisfiable, but $M''$ is consistent.

We extend the definition of $\mathsf{value}$ to clauses so that $\mathsf{value}(C, M) = \mathsf{true}$ if at least one literal of $C$ evaluates to $\mathsf{true}$, $\mathsf{value}(C, M) = \mathsf{false}$ if all literals evaluate to $\mathsf{false}$, and $\mathsf{value}(C, M) = \mathsf{undef}$ otherwise. We say a clause $C$ is satisfied by trail $M$ if $\mathsf{value}(C, M) = \mathsf{true}$. A set of clauses $\mathcal{C}$ is satisfied by $M$ if $M$ is complete (and therefore consistent), and all clauses $C \in \mathcal{C}$ are satisfied by $M$. We use the predicate $\mathsf{satisfied}(\mathcal{C}, M)$ to denote that $\mathcal{C}$ is satisfied by $M$,

Given a set of clauses $\mathcal{C}_0$, our procedure starts with the state $\langle [\![]\!], \mathcal{C}_0 \rangle$ and performs transitions according to the rules we explain below. The goal is to either enter into a state $\mathsf{sat}$ denoting that the problem is satisfiable, or into a state $\mathsf{unsat}$ denoting that the problem is unsatisfiable. The states we traverse are either *search states* of the form $\langle M, \mathcal{C} \rangle$ or *conflict resolution states* of the form $\langle M, \mathcal{C} \rangle \vdash C$. In both types of states we keep the invariant that $M$ is a consistent trail and $\mathcal{C}_0 \subseteq \mathcal{C}$. Additionally, in conflict resolution states the clause $C$ is always a clause implied by $\mathcal{C}_0$ and refuted by the trail, i.e. $\mathcal{C}_0 \vDash C$ and $\mathsf{value}(C) = \mathsf{false}$. We call the clause $C$ the *conflicting clause.*

To ensure termination, the transition system assumes existence of a finite set of literals $\mathbb{B}$ that we call the *finite basis.* During a derivation of the system, any literal added to the trail will be from $\mathbb{B}$, and the clauses that the system uses will only contain literals from $\mathbb{B}$.[4] The minimal assumption is that $\mathbb{B}$ must include all literals (and their negations) from the initial problem $\mathcal{C}_0$, and the theory-specific decision procedure must ensure that for any $\mathcal{C}_0$ such a finite basis exists.

## 2.1 Clausal Rules

We start by presenting the set of search rules and conflict analysis rules that resemble those of abstract DPLL and are the backbone of CDCL-style SAT

[4] Our finite basis corresponds to the closure of the literal-generating function used in splitting-on-demand [1].

solvers. The clausal search rules are presented in Fig. 1 and the clausal conflict analysis rules are presented in Fig. 2.

| | | | | |
|---|---|---|---|---|
| Decide | | | | |
| $\langle M, \mathcal{C} \rangle$ | $\longrightarrow$ | $\langle [\![M, L]\!], \mathcal{C} \rangle$ | **if** | $L \in \mathbb{B}$, $\mathsf{value}(L, M) = \mathsf{undef}$ |
| Propagate | | | | |
| $\langle M, \mathcal{C} \rangle$ | $\longrightarrow$ | $\langle [\![M, C{\rightarrow}L]\!], \mathcal{C} \rangle$ | **if** | $C = (L_1 \vee \ldots \vee L_n \vee L) \in \mathcal{C}$<br>$\forall i : \mathsf{value}(L_i, M) = \mathsf{false}$<br>$\mathsf{value}(L, M) = \mathsf{undef}$ |
| Conflict | | | | |
| $\langle M, \mathcal{C} \rangle$ | $\longrightarrow$ | $\langle M, \mathcal{C} \rangle \vdash C$ | **if** | $C \in \mathcal{C}$, $\mathsf{value}(C) = \mathsf{false}$ |
| Sat | | | | |
| $\langle M, \mathcal{C} \rangle$ | $\longrightarrow$ | sat | **if** | $\mathsf{satisfied}(\mathcal{C}, M)$ |
| Forget | | | | |
| $\langle M, \mathcal{C} \rangle$ | $\longrightarrow$ | $\langle M, \mathcal{C} \setminus \{C\} \rangle$ | **if** | $C \in \mathcal{C}$ is a learned clause. |

**Fig. 1.** Clausal search rules.

*Search Rules.* The Decide rule can take any literal from the basis $\mathbb{B}$ that does not yet have a value added to the trail. The Propagate performs Boolean clause propagation by assigning to true the only unassigned literal from a clause where all other literals already evaluate to false.[5] If we encounter a clause $C$ such all literals in $C$ evaluate to false, we use the Conflict to enter conflict resolution state. During conflict analysis we can learn new clauses, and these can be removed using the Forget rule. If our trail is complete and satisfies all the clauses, we can finish the search using the Sat rule.

*Conflict analysis rules.* As in CDCL, the conflict analysis rules recover from a conflict encountered during the search, learn the reason of the conflict, and backtrack to an appropriate state to continue the search. The main analysis rule is the Resolve rule. This rule performs Boolean resolution of clauses $C$ and $D$ over the literal $L$, obtaining the clause $R = \mathsf{resolve}(C, D, L)$. The clause $R$ is a valid deduction and moreover evaluates to false in $M$. If the result of the resolution is an empty clause (denoted with false), we can deduce that the problem is unsatisfiable using the Unsat rule. Since, in a conflict analysis state $\langle M, \mathcal{C} \rangle \vdash C$, the clause $C$ is a always a valid deduction, we can use the Learn to add the clause $C$ to set of clauses in order to aid in reducing the search space in the future. If this learned clause is at a later point not deemed useful, we can

[5] Note that in both Decide and Propagate we do not need to ensure that the new literal does not cause an infeasibility. Additionally, we can decide literals that do not yet exist in $\mathcal{C}$, enabling refinement decisions as found in [6].

|  |  |  |  |  |
|---|---|---|---|---|
| Resolve |  |  |  |  |
| $\langle [\![M, D{\to}L]\!], \mathcal{C}\rangle \vdash C$ | $\longrightarrow$ | $\langle M, \mathcal{C}\rangle \vdash R$ | **if** | $\neg L \in C$,<br>$R = \mathsf{resolve}(C, D, L)$ |
| Consume |  |  |  |  |
| $\langle [\![M, D{\to}L]\!], \mathcal{C}\rangle \vdash C$ | $\longrightarrow$ | $\langle M, \mathcal{C}\rangle \vdash C$ | **if** | $\neg L \notin C$ |
| $\langle [\![M, L]\!], \mathcal{C}\rangle \vdash C$ | $\longrightarrow$ | $\langle M, \mathcal{C}\rangle \vdash C$ | **if** | $\neg L \notin C$ |
| Backjump |  |  |  |  |
| $\langle [\![M, N]\!], \mathcal{C}\rangle \vdash C$ | $\longrightarrow$ | $\langle [\![M, C{\to}L]\!], \mathcal{C}\rangle$ | **if** | $C = L_1 \vee \ldots \vee L_m \vee L$<br>$\forall i : \mathsf{value}(L_i, M) = \mathsf{false}$<br>$\mathsf{value}(L, M) = \mathsf{undef}$<br>$N$ starts with a decision |
| Unsat |  |  |  |  |
| $\langle M, \mathcal{C}\rangle \vdash \mathsf{false}$ | $\longrightarrow$ | unsat |  |  |
| Learn |  |  |  |  |
| $\langle M, \mathcal{C}\rangle \vdash C$ | $\longrightarrow$ | $\langle M, \mathcal{C} \cup \{C\}\rangle \vdash C$ | **if** | $C \notin \mathcal{C}$ |

**Fig. 2.** Clausal conflict analysis rules.

remove it using the Forget rule. The Consume rules skips over those decided and propagated literals in the trail that are not relevant for the inconsistency of the conflicting clause. Finally, we exit conflict resolution using the Backjump rule, if we have deduced a clause $C$ such that implies a literal earlier in the trail, skipping over at least one decision.

## 2.2 Theory-Specific Rules

We now extend the rules to enable theory-specific reasoning, allowing deductions in the style of DPLL(T), but more flexible, and allowing for assignments of variables to particular concrete values (Figure 3). As in DPLL(T), the basic requirement for a theory decision procedure is to provide an explain function that can explain theory-specific propagations and infeasible states. In DPLL(T), the theory explanations are theory lemmas in form of clauses that only contain negations of literals asserted so far. The explanation function explain here has more flexibility. Given a literal $L$ and a consistent trail $M$ that implies $L$ to be true, i.e. such that $[\![M, \neg L]\!]$ is infeasible, $\mathsf{explain}(L, M)$ must return a valid theory lemma $E = L_1 \vee \ldots L_k \vee L$. The literals of the clause $E$ must be from the finite basis $\mathbb{B}$, and all literals $L_i$ must evaluate to false in $M$. Allowing explanations containing more than just the literals off the trail allows for more expressive lemmas and is crucial for model-based decision procedures. Limiting the literals to a finite basis, on the other hand, is important for ensuring the termination of the procedure.

*Search rules.* We can propagate a literal $L$ using the T-Propagate rule, if it is implied in the current state, i.e. if adding the literal $\neg L$ to the trail $M$ makes

| | | | | |
|---|---|---|---|---|
| T-PROPAGATE | | | | |
| $\langle M, \mathcal{C} \rangle$ | $\longrightarrow$ | $\langle [\![ M, E \rightarrow L ]\!], \mathcal{C} \rangle$ | **if** | $L \in \mathbb{B}$, $\mathsf{value}(L, M) = \mathsf{undef}$<br>$\mathsf{infeasible}([\![ M, \neg L ]\!])$<br>$E = \mathsf{explain}([\![ M, \neg L ]\!])$ |
| T-DECIDE | | | | |
| $\langle M, \mathcal{C} \rangle$ | $\longrightarrow$ | $\langle [\![ M, x \mapsto \alpha ]\!], \mathcal{C} \rangle$ | **if** | $x \in \mathsf{vars}_T(\mathcal{C})$<br>$v[M](x) = \mathsf{undef}$<br>$\mathsf{consistent}([\![ M, x \mapsto \alpha ]\!])$ |
| T-CONFLICT | | | | |
| $\langle M, \mathcal{C} \rangle$ | $\longrightarrow$ | $\langle M, \mathcal{C} \rangle \vdash E$ | **if** | $\mathsf{infeasible}(M)$<br>$E = \mathsf{explain}(\mathsf{false}, M)$ |
| T-CONSUME | | | | |
| $\langle [\![ M, x \mapsto \alpha ]\!], \mathcal{C} \rangle \vdash C$ | $\longrightarrow$ | $\langle M, \mathcal{C} \rangle \vdash C$ | **if** | $\mathsf{value}(C, M) = \mathsf{false}$ |
| T-BACKJUMP-DECIDE | | | | |
| $\langle [\![ M, x \mapsto \alpha, N ]\!], \mathcal{C} \rangle \vdash C$ | $\longrightarrow$ | $\langle [\![ M, L ]\!], \mathcal{C} \rangle$ | **if** | $C = L_1 \vee \ldots \vee L_m \vee L$<br>$\exists i : \mathsf{value}(L_i, M) = \mathsf{undef}$<br>$\mathsf{value}(L, M) = \mathsf{undef}$ |

**Fig. 3.** Theory search and conflict rules.

it infeasible. The side condition $\mathsf{value}(L, M) = \mathsf{undef}$ ensures that we cannot produce an inconsistent trail ($\mathsf{value}(L, M) = \mathsf{false}$), nor include redundant information ($\mathsf{value}(L, M) = \mathsf{true}$). We use the T-DECIDE rule to assign an interpretation/value ($\alpha$) to a variable $x$ that occurs in our set of clauses ($x \in \mathsf{vars}_T(\mathcal{C})$). The side condition $v[M](x) = \mathsf{undef}$ ensures that we cannot assign a variable $x$ that is already assigned in $M$, and $\mathsf{consistent}([\![ M, x \mapsto \alpha ]\!])$ ensures the new trail is consistent. We require a consistent trail because the function $\mathsf{value}$ is not well defined for inconsistent trails. We use the T-CONFLICT rule to enter conflict resolution state, whenever we detect that the trail is infeasible.

*Conflict analysis rules.* The T-CONSUME rule is similar to the consume rules in Figure 2, as it skips over a decided model assignment $x \mapsto \alpha$ that is not relevant for the inconsistency of the conflicting clause. The assignment is not relevant because the conflicting clause $C$ still evaluates to $\mathsf{false}$ after the assignment $x \mapsto \alpha$ is removed from the trail. We use the T-BACKJUMP-DECIDE rule when we reach an assignment $x \mapsto \alpha$ that is relevant for the inconsistency of the conflicting clause $C$, but the BACKJUMP rule is not applicable because $C$ contains more than one literal that evaluates to $\mathsf{undef}$ after we remove the assignment $x \mapsto \alpha$ from the trail. The T-BACKJUMP-DECIDE rule may generates doubts about the termination argument for our abstract procedure, since it is just replacing a decision $x \mapsto \alpha$ with another decision $L$. In our termination proof, we justify that by assuming that Boolean decisions ($L$) have "bigger" weight than model assignment decisions ($x \mapsto \alpha$).

### 2.3 Producing Explanations

A crucial component of our framework is the explanation function $\mathsf{explain}$. Given an infeasible trail $M$, it must be able to produce a valid theory lemma that is inconsistent with $M$ using only literals from the finite basis.

If the infeasibility of $M$ only depends on literals that already occur $M$, then $\mathsf{explain}$ can simply return the inconsistent clause $\neg L_1 \vee \ldots \vee \neg L_k$ where all literal $L_i$ occur in $M$. For example, the trail $M = [\![x < 0,\ y > 1,\ x > 0]\!]$ is infeasible, and $\mathsf{explain}(\mathsf{false}, M) = \neg(x < 0) \vee \neg(x > 0)$ is a valid theory lemma that is inconsistent with $M$. In such cases, the T-Conflict and T-Propagate rules correspond to the theory propagation and conflict rules from the $\mathsf{DPLL(T)}$ framework.

The more interesting case occurs when the infeasibility on a trail $M$ also depends on decided model assignments $x \mapsto \alpha$ in $M$. Consider, for example, the infeasible (but consistent) trail

$$M = [\![y > 0,\ \ z > 0,\ \ x + y + z < 0,\ \ x \mapsto 0]\!] \ .$$

It might be tempting to define $\mathsf{explain}(\mathsf{false}, M)$ to produce the valid theory lemma

$$\neg(y > 0) \vee \neg(z > 0) \vee \neg(x + y + z < 0) \vee \neg(x = 0)$$

This naïve $\mathsf{explain}$ function that just replaces the assignments $x \mapsto \alpha$ with literals $x = \alpha$, is inadequate as it *does not* satisfy the finite basis requirement for theories that operate over infinite domains, such as the Integers or the Reals. Using such a function, in this example, we would be able to continue indefinitely by assigning $x$ to 1, then 2, and so on.

In principle, for any theory that admits elimination of quantifiers, it is possible to construct an explanation function $\mathsf{explain}$ that satisfy the finite basis requirement. The basic idea is to eliminate all unassigned variables and produce an implied formula that is also inconsistent with the assigned variables in the infeasible trail. In the previous example, the variables $y$ and $z$ are unassigned, so we can separate the infeasible literals that the naïve $\mathsf{explain}$ function return into the literals from the trail and literals from assignments

$$A \equiv (y > 0) \wedge (z > 0) \wedge (x + y + z < 0) \qquad\qquad B \equiv (x = 0) \ .$$

Given $A$, we use a quantifier elimination procedure to generate a CNF formula $F$ of the form $C_1 \wedge \ldots \wedge C_k$ that is equivalent to $(\exists y, z : A)$, and therefore also inconsistent with $B$ while only using the assigned variables (in this case $x$). Note that $B$ corresponds to the assignment $v[M]$, and each clause $C_i$ evaluates to $\mathsf{true}$ or $\mathsf{false}$ under $v[M]$ because all variables occurring in $C_i$ are assigned by $v[M]$. Moreover, at least one of these clauses must evaluate to $\mathsf{false}$ because $F$ is inconsistent with $B$. Let $I$ be the clause $C_i$ that evaluates to $\mathsf{false}$. We have that $A \implies I$ is a valid clause because

$$A \implies (\exists y, z : A) \iff F \implies I$$

Since $I$ is inconsistent with $B$, the clause $A \implies I$ will also be inconsistent with the trail and can be used as an explanation. Moreover, $I$ is an *interpolant* for $A$ and $B$. In this example, we obtain $0 < x$ by eliminating the variables $y$ and $z$ from $A$ using Fourier-Motzkin elimination, resulting in the explanation $A \implies (0 < x)$. When solving a set of linear arithmetic $\mathcal{C}$, Fourier-Motzkin elimination is sufficient to define the explain function, as shown in [12,9]. Fourier-Motzkin elimination gives a finite-basis $\mathbb{B}$ with respect to $\mathcal{C}$, and the basis can be obtained by closing $\mathcal{C}$ under the application of Fourier-Motzkin elimination step. It is fairly easy to show that the closure is a finite set, since we always produce constraints with one variable less.

In the last example, $0 < x$ is an *interpolant* for $A$ and $B$, but so is $x \neq 0$. The key point is that an arbitrary interpolation procedure does not guarantee a finite basis. Nonetheless, this observation is useful when designing explanation functions for more complex theories. For nonlinear arithmetic constraints, we describe how to produce an explain procedure that produces an interpolant based on cylindrical algebraic decomposition (CAD) in [8]. The theory of uninterpreted functions (UF) does not admit quantifier elimination, but *dynamic-ackermannization* [5] can be used to create an interpolant $I$ between $A$ and $B$ without compromising the finite basis requirement. For example, the trail $M = [\![x \mapsto 0,\ y \mapsto 0,\ f(x) \neq f(y)]\!]$ is infeasible with respect to UF. Then, we have $A \equiv f(x) \neq f(y)$ and $B \equiv (x = 0) \wedge (y = 0)$. Using dynamic-ackermannization we have that $I \equiv x \neq y$ is an interpolant for $A$ and $B$, and $f(x) \neq f(y) \implies x \neq y$ is a valid theory lemma that is also inconsistent with $M$. We satisfy the finite basis requirement because dynamic-ackermannization only "introduces" equalities between terms that already exist in the trail. Finally, an explain function for the theory of arrays can be built on top of the approach described in [3].

*Example 2.* For the sake of simplicity, we restrict ourselves to the case of linear arithmetic. We illustrate the search rules by applying them to the following clauses over Boolean and linear arithmetic atoms $\mathcal{C} = \{x < 1 \vee p,\ \neg p \vee x = 2\}$. We start the deduction from the initial state $\langle [\![]\!], \mathcal{C}\rangle$ and apply the rules of the mcSAT system.

$\langle [\![]\!], \mathcal{C}\rangle$

$\downarrow$ T-Decide $(x \mapsto 1)$

$\langle [\![x \mapsto 1]\!], \mathcal{C}\rangle$

$\downarrow$ Propagate ($p$ must be true, since $x < 1$ evaluates to false in the current trail)

$\langle [\![x \mapsto 1,\ (x < 1 \vee p) \rightarrow p]\!], \mathcal{C}\rangle$

$\downarrow$ Conflict ($\neg p$ and $x = 2$ evaluate to false in the current trail)

$\langle [\![x \mapsto 1,\ (x < 1 \vee p) \rightarrow p]\!], \mathcal{C}\rangle \vdash \neg p \vee x = 2$

$\downarrow$ Resolve (resolving $x < 1 \vee p$ and $\neg p \vee x = 2$)

$\langle [\![x \mapsto 1]\!], \mathcal{C}\rangle \vdash x < 1 \vee x = 2$

In this state, the Backjump rule is not applicable because in the conflicting clause, both $x < 1$ and $x = 2$ evaluate to undef after the model assignment

$x \mapsto 1$ is removed from the trail. The intuition is that the model assignment was "premature", and the clause $x < 1 \vee x = 2$ is indicating that we should decide $x < 1$ or $x = 2$ before we assign $x$.

$\langle [\![ x \mapsto 1 ]\!], \mathcal{C} \rangle \vdash x < 1 \vee x = 2$
$\downarrow$ T-Backjump-Decide
$\langle [\![ x < 1 ]\!], \mathcal{C} \rangle$
$\downarrow$ T-Decide (the model assignment must satisfy $x < 1$)
$\langle [\![ x < 1, \; x \mapsto 0 ]\!], \mathcal{C} \rangle$
$\downarrow$ Propagate ($\neg p$ must be true, since $x = 2$ evaluates to false in the current trail)
$\langle [\![ x < 1, \; x \mapsto 0, \; (\neg p \vee x = 2) \rightarrow \neg p ]\!], \mathcal{C} \rangle$
$\downarrow$ Sat ($x \mapsto 0$ and $\neg p$ satisfy all clauses)
sat

*Example 3.* Now, we consider a set of linear arithmetic (unit) clauses

$$\mathcal{C} = \{x < 1, \;\; x < y, \;\; 1 < z, \;\; z < x\} \;\; .$$

To simplify the presentation of this example, with a small abuse of notation, we use $\hookrightarrow L$ instead of $L \rightarrow L$, whenever the literal $L$ is implied by the unit clause $L$.

$\langle [\![ ]\!], \mathcal{C} \rangle$
$\downarrow$ Propagate $\times$ 4 (propagate all unit clauses)
$\langle [\![ \hookrightarrow x < 1, \hookrightarrow x < y, \hookrightarrow 1 < z, \hookrightarrow z < x ]\!], \mathcal{C} \rangle$
$\downarrow$ T-Decide (the current trail is consistent with the model assignment $x \mapsto 0$)
$\langle [\![ \hookrightarrow x < 1, \hookrightarrow x < y, \hookrightarrow 1 < z, \hookrightarrow z < x, \; x \mapsto 0 ]\!], \mathcal{C} \rangle$
$\downarrow$ T-Decide (peek a value for $y$, keeping consistency, $y$ s.t. $x < y$)
$\langle [\![ \hookrightarrow x < 1, \hookrightarrow x < y, \hookrightarrow 1 < z, \hookrightarrow z < x, \; x \mapsto 0, \; y \mapsto 1 ]\!], \mathcal{C} \rangle$
$\downarrow$ T-Conflict ($1 < z$ and $z < x$ implies that $1 < x$)
$\langle [\![ \hookrightarrow x < 1, \hookrightarrow x < y, \hookrightarrow 1 < z, \hookrightarrow z < x, \; x \mapsto 0, \; y \mapsto 1 ]\!], \mathcal{C} \rangle \vdash C$

In the state above, the conflict was detected by noticing that we can not pick a value for $z$, because the trail contains $1 < z$, $z < x$ and $x \mapsto 0$. The explain procedure "generates" the explanation clause $C \equiv \neg(1 < z) \vee \neg(z < x) \vee 1 < x$ by eliminating $z$ using Fourier-Motzkin elimination, and this clause evaluates to false in the current trail. Note that, as in DPLL(T), we could have also explained the infeasibility trail by producing the clause $\neg(1 < z) \vee \neg(z < x) \vee \neg(x < 1)$ that uses only literals that already exist in the trail. However, this is clause is weaker since $\neg(x < 1) \equiv (1 \leq x)$ is weaker than $1 < x$. We continue the deduction by analyzing the conflict.

$\langle [\![ \hookrightarrow x < 1, \hookrightarrow x < y, \hookrightarrow 1 < z, \hookrightarrow z < x, \; x \mapsto 0, \; y \mapsto 1 ]\!], \mathcal{C} \rangle \vdash C$
$\downarrow$ T-Consume (the conflict does not depend on $y$)
$\langle [\![ \hookrightarrow x < 1, \hookrightarrow x < y, \hookrightarrow 1 < z, \hookrightarrow z < x, \; x \mapsto 0 ]\!], \mathcal{C} \rangle \vdash C$
$\downarrow$ Backjump (after backtracking $x \mapsto 0$, the clause $C$ implies $1 < x$)
$\langle [\![ \hookrightarrow x < 1, \hookrightarrow x < y, \hookrightarrow 1 < z, \hookrightarrow z < x, \; C \rightarrow 1 < x ]\!], \mathcal{C} \rangle$

After the application of the backjump rule, the newly asserted literal $1 < x$ is immediately in conflict with the literal $x < 1$ and we enter conflict resolution again, with the explanation obtained by Fourier-Motzkin elimination.

$\langle [\![ \rightarrowtail x < 1, \rightarrowtail x < y, \rightarrowtail 1 < z, \rightarrowtail z < x,\ C{\to}1 < x ]\!], \mathcal{C} \rangle$

$\downarrow$ T-CONFLICT $(1 < x$ and $x < 1$ implies $\mathsf{false})$

$\langle [\![ \rightarrowtail x < 1, \rightarrowtail x < y, \rightarrowtail 1 < z, \rightarrowtail z < x,\ C{\to}1 < x ]\!], \mathcal{C} \rangle \vdash \neg(1 < x) \vee \neg(x < 1)$

$\downarrow$ RESOLVE $\times\, 3$, CONSUME, RESOLVE, UNSAT

unsat

**Theorem 1.** *Given a set of clauses $\mathcal{C}$, and assuming a finite basis explanation function* explain*, any derivation starting from the initial state $\langle [\![]\!], \mathcal{C}\rangle$ will terminate either in a state* sat*, when $\mathcal{C}$ is satisfiable, or in the* unsat *state. In the later case, the set of clauses $\mathcal{C}$ is unsatisfiable.*

*Proof.* Assume we have a set of clauses $\mathcal{C}$, over the variables $x_1, \ldots, x_n$, and a finite-basis explanation function explain. Starting from the initial state $\langle [\![]\!], \mathcal{C}\rangle$, we claim that any derivation of the transition system (finite or infinite), satisfies the following properties

1. the only possible "sink states" are the sat and the unsat states;
2. all $\vdash C$ clauses are implied by the initial set of clauses $\mathcal{C}$;
3. during conflict analysis $\mathsf{value}(C, M) = \mathsf{false}$ for the $\vdash C$ clauses;

Assuming termination, and the above properties, the statement can be proven easily. Since sat and unsat are the only sink states, the derivation will terminate in one of these states. Since the SAT rule is only applicable if the set of clauses $\mathcal{C}$ is satisfied, we have that the original problem is indeed satisfiable. On the other hand, if we terminate in the unsat state, by the above properties, the conflicting clause false is implied by the inital $\mathcal{C}$. Given that false is implied by the original set of clauses, the initial clauses themselves must truly be unsatisfiable.

The first property in the list above is a fairly easy exercise in case analysis and induction, so we skip those and concentrate on the more interesting properties. Proving the properties of conflict analysis is also quite straightforward, via induction on the number of conflicts, and conflict analysis steps. Clearly, initially, we have that $C$ evaluates to false (the precondition of the CONFLICT and T-CONFLICT rules), and is implied by $\mathcal{C}$ by induction. Then, every new clause that we produce during conflict resolution is obtained by the Boolean resolve rule, which will produce a valid deduction. Additionally, since the clause we are resolving with is a proper explanation, it will have all literals except the one we are resolving evaluate to false. Therefore, the resolvent also evaluates to false. As we backtrack down the trail with the conflicting clause, by definition of value and the preconditions of the rules, the clause still remains false.

Now, let us prove that the system terminates. It is clear that the conflict analysis rules (always removing elements from the trail) always terminate in a finite number of steps, and return to the search rules (or the unsat state). For the sake of the argument, let us assume that there is a derivation that does not terminate, and therefore does not enter the $\mathsf{unsat}$ state. We can define a big-step transition relation $\longrightarrow_{\mathrm{bs}}$ that covers a transition from a search state, applying one or more transitions in the conflict analysis rules, and returns to a search state.

By assumption, we have a finite-basis explanation function $\mathsf{explain}$, so we can assume a set of literals $\mathbb{B}$ from which all the clauses that we can see during the search are constructed. In order to keep progress of the search, we define a partial order $M_1 \prec M_2$ on trails. The trail contains three different kinds of element: model assignments decisions $(x \mapsto \alpha)$, Boolean decisions $(L)$ and propagations $(C \rightarrow L)$. The basic idea is to consider that propagations are heavier than Boolean decisions that are heavier than model assignments. We capture that by defining a (weight) function $w$ from trail elements into the set $\{1, 2, 3\}$.

$$w(C \rightarrow L) = 3, \quad w(L) = 2, \quad w(x \mapsto \alpha) = 1$$

We define the $M_1 \prec M_2$ using a lexicographical order based on the weights of the trail elements, i.e.

$$\begin{aligned} \llbracket\rrbracket \prec M &= \mathsf{true}\ \textbf{if}\ M \neq \llbracket\rrbracket \\ M \prec \llbracket\rrbracket &= \mathsf{false} \\ \llbracket a, M_1 \rrbracket \prec \llbracket b, M_2 \rrbracket &= w(a) < w(b) \lor (w(a) = w(b) \land M_1 \prec M_2) \end{aligned}$$

It is clear that $\llbracket\rrbracket$ is the minimal element, and any trail containing $|\mathbb{B}|$ propagations followed by $n$ model assignments is maximal. It is easy to see that for all trails $M$ and trail elements $a$ we have that $M \prec \llbracket M, a \rrbracket$. Thus, any rule that adds a new element to the trail is essentially creating a "bigger" trail with respect to the partial order $\prec$. This simple observation covers most of our rules.

Now, we consider the big-step $\longrightarrow_{\mathrm{bs}}$ transition from a state $\langle M_1, \mathcal{C}_1 \rangle$ into a state $\langle M_2, \mathcal{C}_2 \rangle$. If we return using Backjump, then the trail $M_1$ is of the form $\llbracket M, L, N \rrbracket$ or $\llbracket M, x \mapsto \alpha, N \rrbracket$, and the trail $M_2$ is of the form $\llbracket M, C \rightarrow L' \rrbracket$. In both cases $M_1 \prec M_2$ because a $w(C \rightarrow L')$ is greater than $w(L)$ and $w(x \mapsto \alpha)$. Similarly, if we return using T-Backjump-Decide, the trail $M_1$ is of the form $\llbracket M, x \mapsto \alpha, N \rrbracket$, and $M_2$ is of the form $\llbracket M, L \rrbracket$, once again $M_1 \prec M_2$ since $w(L) > w(x \mapsto \alpha)$.

Now, to justify the Forget rule, we define a partial order $\langle M_1, \mathcal{C}_1 \rangle \lessdot \langle M_2, \mathcal{C}_2 \rangle$ as $M_1 \prec M_2 \lor (M_1 = M_2 \land |\mathcal{C}_1| > |\mathcal{C}_2|)$. Using this definition, we have that $\langle M, \mathcal{C} \rangle \lessdot \langle M, \mathcal{C} \setminus \{C\} \rangle$, and consequently the Forget rule also produces a "bigger" state. The partial order $\lessdot$ also have maximal elements $\langle M_{\mathsf{max}}, \mathcal{C}_0 \rangle$, where $M_{\mathsf{max}}$ is a maximal element for $\prec$ and $\mathcal{C}_0$ is the initial set of clauses. Since all rules are producing bigger states and we can not increase forever, the termination of the system follows. $\square$

## 3 Conclusion

We proposed a model-constructing satisfiability calculus ($\mathsf{mcSAT}$) that encompasses all model-based decision procedures found in the SMT literature. The mcSAT framework extends DPLL(T) by allowing assignments of variables to concrete values and relaxing the restriction on creation of new literals. The model created during the search also takes part in explaining the conflicts, and the full model is readily available as a witness if the procedure reports a satisfiable answer. The new calculus also extends nlsat, proposed at [8], by removing unnecessary restrictions on the Boolean-level search rules – it allows efficient Boolean constraint propagation found in state-of-the-art SAT solvers and incorporates theory propagation and conflict rules from the DPLL(T) framework. The mcSAT calculus allows SMT developers to combine existing and successful techniques from the DPLL(T) framework with the flexibility provided by the nlsat calculus. We also presented a correctness proof for our procedure that is much simpler than the one provided for nlsat.

In this article, we did not explore the theory combination problem. However, we believe mcSAT is the ideal framework for combining implementations for theories such as: arithmetic, bit-vectors, floating-point, uninterpreted functions, arrays and datatypes.

## References

1. C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Splitting on demand in SAT modulo theories. In *LPAR*, pages 512–526, 2006.
2. C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability.* IOS Press, 2009.
3. R. Brummayer and A. Biere. Lemmas on Demand for the Extensional Theory of Arrays. *Journal on Satisfiability, Boolean Modeling and Computation*, 6, 2009.
4. S. Cotton. Natural domain SMT: A preliminary assessment. In *FORMATS*, 2010.
5. L. de Moura and N. Bjørner. Model-based Theory Combination. In *Satisfiability Modulo Theories*, volume 198 of *ENTCS*, pages 37–49, 2008.
6. L. Haller, A. Griggio, M. Brain, and D. Kroening. Deciding floating-point logic with systematic abstraction. In *FMCAD*, 2012.
7. D. Jovanović and L. de Moura. Cutting to the chase: Solving linear integer arithmetic. In *Automated Deduction*, pages 338–353. Springer, 2011.
8. D. Jovanović and L. de Moura. Solving non-linear arithmetic. *Automated Reasoning*, pages 339–354, 2012.
9. K. Korovin, N. Tsiskaridze, and A. Voronkov. Conflict resolution. *Principles and Practice of Constraint Programming*, pages 509–523, 2009.
10. S. Krstić and A. Goel. Architecting solvers for SAT modulo theories: Nelson-Oppen with DPLL. *Frontiers of Combining Systems*, pages 1–27, 2007.
11. S. Malik and L. Zhang. Boolean satisfiability from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76–82, 2009.
12. K. L. McMillan, A. Kuehlmann, and M. Sagiv. Generalizing DPLL to richer logics. In *Computer Aided Verification*, pages 462–476. Springer, 2009.
13. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *DAC*, 2001.
14. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL($T$). *Journal of the ACM*, 53(6):937–977, 2006.
15. J. P. M. Silva and K. A. Sakallah. GRASP – a new search algorithm for satisfiability. In *ICCAD*, 1997.