

Architecting Solvers for SAT Modulo Theories: Nelson-Oppen with DPLL

Sava Krstić and Amit Goel

Strategic CAD Labs, Intel Corporation

Abstract. We offer a transition system representing a high-level but detailed architecture for SMT solvers that combine a propositional SAT engine with solvers for multiple disjoint theories. The system captures succinctly and accurately all the major aspects of the solver’s global operation: boolean search with across-the-board backjumping, communication of theory-specific facts and equalities between shared variables, and cooperative conflict analysis. Provably correct and prudently underspecified, our system is a readily usable ground for high-quality implementations of comprehensive SMT solvers.

1 Introduction

SMT solvers are fully automated theorem provers based on decision procedures. The acronym is for *Satisfiability Modulo Theories*, indicating that an SMT solver works as a satisfiability checker, with its decision procedures targeting queries from one or more logical theories. These proof engines have become vital in verification practice and hold an even greater promise, but they are still a challenge to design and implement. From the seminal *Simplify* [8] to the current state-of-the-art *Yices* [9], with notable exceptions such as *UCLID* [5], the prevailing wisdom has been that an SMT solver should contain a SAT solver for managing the boolean complexity of the input formula and several specialized solvers—linear arithmetic and “theory of uninterpreted functions” obbligate—that communicate by exchanging equalities between variables (“the Nelson-Oppen style” [12]). This much granted, there is a host of remaining design issues at various levels of abstraction, the response to which distinguishes one solver from another.

Our goal is to define the top-level architecture of an SMT solver as a mathematical object that can be grasped as a whole and fruitfully reasoned about. We want an abstract model that faithfully captures the intricacies of the solver’s global operation—what is going on between the architectural components and what is going on inside the components that is essential for interaction. We achieve this goal by presenting the SMT solver as a non-deterministic transition system. The ten rules of our system (Figure 4) provide a rather detailed rational reconstruction of the mainstream SMT solvers, covering the mechanisms for boolean search, communication of theory-specific facts and equalities between shared variables, and global conflict analysis. The system provides a solid theoretical basis for implementations, which can explore various execution

strategies, refinements and optimizations, assured of fundamental correctness as long as they “play by the rules”.

Following the precursor [10] to this paper, we adopt a logic with parametric polymorphism as the natural choice for SMT solvers, emphasizing *cardinality constraints*—not the traditional *stable-infinity condition*—as an accurate expression of what matters for completeness of the Nelson-Oppen method in practice. Our main results are the termination and soundness theorems for our transition system. We can also prove completeness (relying on [10]) under well-known assumptions, but stronger completeness results are yet to be found.

Related Work. We were inspired mainly by the work of Nieuwenhuis, Oliveras, and Tinelli [13] on *abstract DPLL* and *abstract DPLL modulo theories*—transition systems that model a DPLL-style SAT solver [7] and an SMT solver that extends it with a solver for *one* theory. In the follow-up paper [3], the same authors with Barrett extend their system with features for “splitting on demand” and derive from it the *DPLL(T_1, \dots, T_n) architecture*. This architecture is closely related to our system NODPLL (Section 4.2), but is significantly less detailed and transparent. It refines DPLL *modulo a single (composite) theory* with appropriate purity requirements on some, but not all rules. In contrast, NODPLL is explicitly *modulo multiple theories*, with rules specifying actions of specific theory solvers and the solvers’ interaction made vivid. For example, equality propagation is spelled out in NODPLL, but which solver in *DPLL(T_1, \dots, T_n)* derives $x = z$ from $x = y$ and $y = z$ is not clear. Another important difference is in the modeling of conflict analysis and it shows even if our systems are compared at the propositional (SAT solver) level. While [13] and [3] view conflict analysis abstractly, tucking it in a general rule for backjumping, NODPLL has rules that directly cover its key steps: conflict detection, the subsequent sequence of “explanations”, generation of the “backjump clause”, and the actual backjump. In an SMT solver, in particular with multiple theories, conflict analysis is even more subtle than in a SAT solver, and the authors of [13] are the first to point out its pitfalls (“too new explanations”) and identify a condition for its correct behavior. NODPLL neatly captures this condition as a guard of a rule.

Our work also builds on [6], which has a transition system modeling a Nelson-Oppen solver for multiple theories, but does not address the cooperation with the SAT solver. Formal models of SMT solvers that do handle a SAT solver together with more than one theory are given only in the paper [3] discussed above and earlier works [2], [4]. Barrett’s architecture of *CVC Lite* as described in [2] is complex and too low-level for convenient analysis and application. The system *SMT($T_1 \cup T_2$)* of Bozzano et al. [4] describes in pseudo-code a particular approach for equality propagation taken by the *MathSAT* solver, which can be modeled in NODPLL; see Section 4.4.

Outline. Section 2 contains (terminological) background as developed in [10], but divorcing the solver’s polymorphic language from *HOL*, to emphasize that parametricity is not tied to higher-order logic, even though it is most conveniently expressed there. In Section 3, we overview purification—a somewhat involved

$$\begin{aligned}
\Sigma_{\text{Eq}} &= \langle \mathbf{Bool} \mid =^{\alpha^2 \rightarrow \text{Bool}}, \text{ite}^{[\text{Bool}, \alpha, \alpha] \rightarrow \alpha}, \text{true}^{\text{Bool}}, \text{false}^{\text{Bool}}, \neg^{\text{Bool} \rightarrow \text{Bool}}, \wedge^{\text{Bool}^2 \rightarrow \text{Bool}}, \dots \rangle \\
\Sigma_{\text{UF}} &= \langle \Rightarrow \mid @^{[\alpha \Rightarrow \beta, \alpha] \rightarrow \beta} \rangle \\
\Sigma_{\text{Int}} &= \langle \text{Int} \mid 0^{\text{Int}}, 1^{\text{Int}}, (-1)^{\text{Int}}, \dots, +^{\text{Int}^2 \rightarrow \text{Int}}, -^{\text{Int}^2 \rightarrow \text{Int}}, \times^{\text{Int}^2 \rightarrow \text{Int}}, \leq^{\text{Int}^2 \rightarrow \text{Bool}}, \dots \rangle \\
\Sigma_{\times} &= \langle \times \mid \langle -, - \rangle^{[\alpha, \beta] \rightarrow \alpha \times \beta}, \text{fst}^{\alpha \times \beta \rightarrow \alpha}, \text{snd}^{\alpha \times \beta \rightarrow \beta} \rangle \\
\Sigma_{\text{Array}} &= \langle \mathbf{Array} \mid \text{mk_arr}^{\beta \rightarrow \text{Array}(\alpha, \beta)}, \text{read}^{[\text{Array}(\alpha, \beta), \alpha] \rightarrow \beta}, \text{write}^{[\text{Array}(\alpha, \beta), \alpha, \beta] \rightarrow \text{Array}(\alpha, \beta)} \rangle \\
\Sigma_{\text{List}} &= \langle \mathbf{List} \mid \text{cons}^{[\alpha, \text{List}(\alpha)] \rightarrow \text{List}(\alpha)}, \text{nil}^{\text{List}(\alpha)}, \text{head}^{[\text{List}(\alpha), \alpha] \rightarrow \text{Bool}}, \text{tail}^{[\text{List}(\alpha), \text{List}(\alpha)] \rightarrow \text{Bool}} \rangle
\end{aligned}$$

Fig. 1. Signatures for theories of some familiar datatypes. For space efficiency, the constants’ arities are shown as superscripts. Σ_{Eq} contains the type operator **Bool** and standard LOGICAL CONSTANTS. All other signatures by definition contain Σ_{Eq} , but to avoid clutter we leave their Σ_{Eq} -part implicit. In Σ_{UF} , the symbol UF is for *uninterpreted functions* and the intended meaning of @ is the function application. The list functions *head* and *tail* are partial, so are represented as predicates in Σ_{List} .

procedure in the context of parametric theories—and give a suitable form of the non-deterministic Nelson-Oppen combination theorem of [10]. Section 4 presents our transition systems, main results, and some discussion. All proofs are given in the appendix.

2 Preliminaries

We are interested in logical theories of common datatypes and their combinations (Figure 1). A datatype has its syntax and semantics; both are needed to define the theory of the datatype. We give an extremely brief overview of the syntax and an informal sketch of semantics, referring to [10] for technical details.

Types A set O of symbols called TYPE OPERATORS, each with an associated non-negative arity, and an infinite set of TYPE VARIABLES define the set Tp_O of TYPES over O . It is the smallest set that contains type variables and expressions $F(\sigma_1, \dots, \sigma_n)$, where $F \in O$ has arity n and $\sigma_i \in \text{Tp}_O$.

A TYPE INSTANTIATION is a finite map from type variables to types. For any type σ and type instantiation $\theta = [\sigma_1/\alpha_1, \dots, \sigma_n/\alpha_n]$, $\theta(\sigma)$ denotes the simultaneous substitution of every occurrence of α_i in σ with σ_i . We say that τ is an INSTANCE of σ and write $\tau \sqsubseteq \sigma$ if there is some θ such that $\tau = \theta(\sigma)$.

Signatures A SIGNATURE is a pair $\langle O \mid K \rangle$, where O is a set of type operators and K is a set of CONSTANTS typed over O . By this we mean that every element of K has an ARITY, which is a tuple of types $(\sigma_0, \dots, \sigma_n)$. Here, $\sigma_1, \dots, \sigma_n$ are the argument types of k , and σ_0 is its range type. We will use the more intuitive notation $k :: [\sigma_1, \dots, \sigma_n] \rightarrow \sigma_0$ to indicate the arity of a constant. Moreover, we will write $k : [\tau_1, \dots, \tau_n] \rightarrow \tau_0$ if there is a type instantiation that maps $\sigma_0, \dots, \sigma_n$ to τ_0, \dots, τ_n respectively. Note the use of $::$ and $:$ for the “principal type” and “type instance” of k respectively. Also note that arities are not types—the symbol \rightarrow is not a type operator. Constants whose range type is **Bool** will be called PREDICATES.

Terms For a given signature $\Sigma = \langle O \mid K \rangle$, and every $\sigma \in \mathsf{Tp}_O$, we assume there is an infinite set of *variables of type* σ ; we write them in the (name,type)-form v^σ . The sets Tm_σ of Σ -TERMS OF TYPE σ are defined inductively by these rules:

- (1) every variable v^σ is in Tm_σ
- (2) if $t_1 \in \mathsf{Tm}_{\tau_1}, \dots, t_n \in \mathsf{Tm}_{\tau_n}$ and $k: [\tau_1, \dots, \tau_n] \rightarrow \tau_0$, then $k t_1 \dots t_n \in \mathsf{Tm}_{\tau_0}$

Type instantiations act on terms: define $\theta(t)$ to be the term obtained by replacing every variable x^σ in t with $x^{\theta(\sigma)}$. If $t \in \mathsf{Tm}_\sigma$, then $\theta(t) \in \mathsf{Tm}_{\theta(\sigma)}$. We say that t' is an INSTANCE of t , written $t' \sqsubseteq t$, if $t' = \theta(t)$ for some θ .

Semantics The type operators `List` and `Array` have arities one and two respectively. The meaning of `List` is a function of arity one (by abuse of notation, also denoted `List`) that given a set E as an argument produces the set `List`(E) of all lists with elements in E . The meaning of `Array` is a function that given two sets I and E as arguments produces the set `Array`(I, E) of arrays indexed by I with elements in E . These functions are *parametric*: there is a certain uniformity in their definition—easily perceived but not easy to pin down precisely. See [10].

The meaning of polymorphic types is defined once we know the meaning of type operators. For example, the meaning of the type `Array`($\alpha, \mathsf{Array}(\alpha, \beta)$) is a function that given any two sets I and E (as interpretations of type variables α, β) produces the set `Array`($I, \mathsf{Array}(I, E)$). If there are no occurrences of type variables in a type (e.g., `List`(`Bool` \times `Int`)), then the meaning of that type is always the same set; if the set is finite, we call the type FINITE.

The meaning of a constant is an indexed family of functions. For example, the meaning of `cons` is the family $\{\mathsf{cons}_E \mid E \text{ is a set}\}$, where cons_E is a function that takes an argument in E and an argument in `List`(E) and produces a result in `List`(E). The family $\{\mathsf{cons}_E\}$ is also *parametric*: see [10] again for a precise expression of the uniformity perceived here.

The meanings of type operators and constants of a signature together determine a STRUCTURE for that signature. The structure gives meaning to all terms. Consider $t = \mathsf{read}(\mathsf{write}(a^{\mathsf{Array}(\alpha, \beta)}, i^\alpha, x^\beta), j^\alpha)$. Once α and β are interpreted as concrete sets (I and E , say) and interpretations for the variables a, i, x, j (elements of `Array`(I, E), I, E, I respectively) are given, the polymorphic term t becomes a well-defined element of E .

We should note that there is more than one reasonable definition of parametricity. For our purposes, it suffices to know that there are structures $\mathcal{T}_{\mathsf{Eq}}, \dots, \mathcal{T}_{\mathsf{List}}$ corresponding to signatures in Figure 1, and that they are all *fully parametric* in the sense of [10] except for $\mathcal{T}_{\mathsf{UF}}$. The structures describing sets, multisets, and arbitrary algebraic datatypes are also fully parametric.

Two signatures are DISJOINT if the only type operators and constants they share are those of Σ_{Eq} . If $\mathcal{T}_1, \dots, \mathcal{T}_n$ are structures with pairwise disjoint signatures, then there is a well-defined *sum structure* $\mathcal{T} = \mathcal{T}_1 + \dots + \mathcal{T}_n$; the semantics of its type operators and constants is defined by the structures they come from. The types and terms of each \mathcal{T}_i are types and terms of \mathcal{T} too. We will call them PURE, or i -PURE when we need to be specific. The attribute MIXED will be used for arbitrary terms and types of a sum structure.

Satisfiability A Σ -FORMULA is an element of $\mathsf{Tm}_{\mathsf{Bool}}$. If ϕ is a Σ -formula and \mathcal{T} is a Σ -structure, we say that ϕ is SATISFIABLE in \mathcal{T} if the meaning of ϕ is true for some interpretations of type variables and ordinary variables occurring in ϕ . We also say that ϕ is VALID if $\neg\phi$ is unsatisfiable. Validity is denoted $\models_{\mathcal{T}} \phi$, and $\phi_1, \dots, \phi_n \models_{\mathcal{T}} \phi$ is an abbreviation for $\models_{\mathcal{T}} \phi_1 \wedge \dots \wedge \phi_n \supset \phi$. The THEORY of a structure is the set of formulas that are valid in it.

An ATOMIC Σ -FORMULA is either a propositional variable or a term of the form $k t_1 \dots t_n$, where k is a predicate. A Σ -LITERAL is an atomic formula or its negation. A CLAUSE is a disjunction of literals. (Clauses containing the same literals in different order are considered equal.) A CONVEX THEORY is defined by the property that if a set of literals implies a disjunction of equalities, then one of the disjuncts must be implied.

A CARDINALITY CONSTRAINT is an “equality” of the form $\sigma \doteq n$, where n is a positive integer; any assignment of sets to type variables occurring in σ makes this constraint either true or false.

Solvers A SOLVER for a fragment of a theory is a sound and complete satisfiability checker for sets of formulas (QUERIES) in the fragment. A STRONG SOLVER checks satisfiability of queries that contain formulas and cardinality constraints.

Normally, theory solvers are built for queries consisting of literals only. The well-known argument that this is sufficient in general begins with the observation that every query Φ is equisatisfiable with one of the form $Q = \Phi_0 \cup \{p_1 \Leftrightarrow \phi_1, \dots, p_n \Leftrightarrow \phi_n\}$, where the p_i are propositional variables, the ϕ_i are literals, and Φ_0 is a propositional query, the boolean skeleton of Φ . A truth assignment M to propositional variables that satisfies Φ_0 can be extended to a model for Φ if and only if the query of literals $Q_M = \{\phi'_1, \dots, \phi'_n\}$ is \mathcal{T} -satisfiable, where ϕ'_i is either ϕ_i or $\neg\phi_i$, depending on whether $M(p_i)$ is true or false. Thus, satisfiability of Φ is decided by checking if Q_M is satisfiable for some model M of Φ_0 . This, of course, calls for a SAT solver to efficiently enumerate the models M .

3 Purification and Non-Deterministic Nelson-Oppen

In the untyped setting, to *purify* a query consisting of mixed formulas is to transform it into an equisatisfiable query consisting of pure formulas. The transformation iteratively replaces a pure subterm t in a mixed formula with a fresh proxy variable x , and adds the pure *definitional equality* $x = t$ to the query. For example, purifying the one-formula query $\{1 + f(x) = f(1 + f(x))\}$ results in the query $\{y = f(x), z = 1 + y, u = f(z), z = u\}$ that contains two pure *UF*-equalities, one pure arithmetical equality, and one equality between variables, which is a pure formula in any theory. (For Σ_{UF} -terms, we use the familiar notations $f(x)$ or $f x$ instead of the syntactically correct $f@x$.) The essence of Nelson-Oppen cooperation is in giving each theory solver the part of the purified query that it understands and then proceed with the solvers deducing in turn new equalities between variables and letting the other solvers know about them.

Types complicate purification, exposing the pertinence of *cardinality constraints*. The typed version of the example above would have $\{y^{\mathsf{Int}} = f^{\mathsf{Int} \Rightarrow \mathsf{Int}} x^{\mathsf{Int}},$

$u^{\text{Int}} = f^{\text{Int} \Rightarrow \text{Int}} z^{\text{Int}}$ as a pure query to pass to the solver for uninterpreted functions. But this query is not pure since the type Int is foreign to the theory \mathcal{T}_{UF} . As a way out, we can create the “type-abstracted” modification $\{y^\alpha = f^{\alpha \Rightarrow \alpha} x^\alpha, u^\alpha = f^{\alpha \Rightarrow \alpha} z^\alpha\}$, which *is* a pure \mathcal{T}_{UF} -query; however, while being sound, this transformation may compromise completeness. Take the example

$$\begin{aligned} \Phi_1 &: \text{distinct}(\text{fst } x^{\text{Int} \times \text{Int}}, \text{snd } y^{\text{Int} \times \text{Int}}, \text{fst } z^{\text{Int} \times \text{Int}}) \\ \Phi_2 &: \text{distinct}(\text{fst } x^{\text{Bool} \times \text{Bool}}, \text{snd } y^{\text{Bool} \times \text{Bool}}, \text{fst } z^{\text{Bool} \times \text{Bool}}) \\ \Phi &: \text{distinct}(\text{fst } x^{\alpha \times \beta_1}, \text{snd } y^{\beta_2 \times \alpha}, \text{fst } z^{\alpha \times \beta_3}) \end{aligned} \quad (1)$$

where $\text{distinct}(x_1, \dots, x_n)$ denotes the query consisting of all disequalities $x_i \neq x_j$. The \mathcal{T}_x -query Φ is the best pure approximation for both Φ_1 and Φ_2 . It is satisfiable, and so is the $(\mathcal{T}_x + \mathcal{T}_{\text{Int}})$ -query Φ_1 , but the $(\mathcal{T}_x + \mathcal{T}_{\text{Bool}})$ -query Φ_2 is *not* satisfiable. Thus, to make a \mathcal{T}_x -solver properly deal with Φ_2 , we should give it the abstraction Φ together with the cardinality constraint $\alpha \doteq 2$.

3.1 Semipurity

Let us now repeat the above in general terms. Suppose $\Sigma = \Sigma_1 + \dots + \Sigma_n$ is a sum of pairwise disjoint signatures $\Sigma_i = \langle O_i \mid K_i \rangle$ and $\mathcal{T} = \mathcal{T}_1 + \dots + \mathcal{T}_n$ is the corresponding sum of theories. A Σ -term will be called *i*-SEMPURE if it can be generated using only constants from K_i . A term is STRICTLY *i*-SEMPURE if it is *i*-semipure and has no occurrences of logical constants. The “impurity” of semipure terms is thus restricted to the type level. For example, the queries Φ_1 and Φ_2 above are semipure (i.e. all their formulas are semipure), while Φ is pure.

Every semipure term t has the best (most general) PURE APPROXIMATION t^{pure} characterized by: $t \sqsubseteq t^{\text{pure}}$ and $t' \sqsubseteq t^{\text{pure}}$ for every pure term t' such that $t \sqsubseteq t'$. The term t^{pure} is unique up to renaming of type variables and can be obtained by erasing all type information from t and then applying a type inference algorithm. (For type inference, see, e.g., [11].) By an easy generalization, for every semipure query Φ there exists an essentially unique pure approximation Φ^{pure} . For example, $\Phi_1^{\text{pure}} = \Phi_2^{\text{pure}} = \Phi$, where the queries are from (1).

If the query Φ is semipure and θ is a type instantiation such that $\theta(\Phi^{\text{pure}}) = \Phi$, denote by Φ^{card} the set of cardinality constraints $\alpha \doteq n$, where $\alpha \in \text{dom}(\theta)$ and $\theta(\alpha)$ is a finite type of cardinality n . One can show that *an i-semipure query Φ is \mathcal{T} -satisfiable if and only if $\Phi^{\text{pure}} \cup \Phi^{\text{card}}$ is \mathcal{T}_i -satisfiable*. (See appendix, Lemma 1.) Thus, a strong solver for *i*-pure queries suffices to solve *i*-semipure queries.

3.2 Purification

Let \mathcal{T} and $\mathcal{T}_1, \dots, \mathcal{T}_n$ be as above. Given any input \mathcal{T} -query, we can purify it by introducing proxy variables and definition equalities and so obtain an equisatisfiable \mathcal{T} -query, say Φ , that contains only semipure formulas. In fact, we can

obtain Φ such that every formula in it is either a propositional clause or has one of the following DEFINITIONAL FORMS

$$(A) \ p \Leftrightarrow (x = y) \quad (B) \ p \Leftrightarrow \phi \quad (C) \ x = t$$

where: (i) p, x, y are variables, ϕ is a strictly semipure literal, and t is a strictly semipure term; (ii) every variable occurs as the left-hand side in at most one definitional form; (iii) no (propositional) variable that occurs in propositional clauses of Φ can occur in the right-hand side of any definitional form. This is proved in the appendix (Lemma 3) under the assumption that non-logical constants do not take arguments of boolean type.¹

Partition now Φ into subsets $\Phi_{\mathbb{B}}, \Phi_{\mathbb{E}}, \Phi_1, \dots, \Phi_n$, where $\Phi_{\mathbb{B}}$ contains the propositional formulas in Φ , $\Phi_{\mathbb{E}}$ contains definitional forms (A), and each Φ_i contains the definitional forms (B) and (C) whose right-hand sides ϕ, t are strictly i -semipure. Note that $\Phi_{\mathbb{B}}$ is i -pure for every i , and $\Phi_{\mathbb{E}}$ is i -semipure for every i .

Example 1. Purifying the formula $f(x) = x \wedge f(2x - f(x)) > x$ produces $\Phi_{\mathbb{B}} = \{p, q\}$, $\Phi_{\mathbb{E}} = \{p \Leftrightarrow y = x\}$, $\Phi_{\text{UF}} = \{y = f(x), u = f(z)\}$, $\Phi_{\text{Int}} = \{z = 2x - y, q \Leftrightarrow u > x\}$. For readability we omit type superscripts on variables; it is clear that p, q are of type `Bool` and x, y, z, u are of type `Int`.

3.3 Nelson-Oppen

Let $\Phi = \Phi_{\mathbb{B}} \cup \Phi_{\mathbb{E}} \cup \Phi_1 \cup \dots \cup \Phi_n$ be as above and let V be the set of all variables occurring in Φ . An ARRANGEMENT on V is a consistent query that for every two variables $x, y \in V$ of the same type contains either $x = y$ or $x \neq y$. Partitioning V into subsets V^σ according to the types of variables, we see that an arrangement determines and is determined by a set of equivalence relations on each class V^σ .

The following result is a slightly more general version of (and easily derived from) Theorem 1 of [10]. It is the basis of the non-deterministic Nelson-Oppen procedure in the style of Tinelli-Harandi [14], but for parametric theories.

Theorem 1. *Suppose all theories \mathcal{T}_i are flexible in the sense of [10].² Let M be an assignment³ to propositional variables occurring in $\Phi_{\mathbb{B}}$ and let Δ be an arrangement of all remaining variables occurring in Φ . Then: $\Phi \cup \Delta \cup M$ is \mathcal{T} -satisfiable if and only if: (1) $M \models \Phi_{\mathbb{B}}$, (2) $M, \Delta \models \Phi_{\mathbb{E}}$, and (3) $(\Phi_i \cup \Delta)^{\text{pure}} \cup \Phi_i^{\text{card}} \cup M$ is \mathcal{T}_i -satisfiable for every $i = 1, \dots, n$. \square*

¹ This assumption is hardly a restriction since we have the polymorphic if-then-else constant `ite` handy in Σ_{Eq} . Bringing the input query to the desired equisatisfiable form Φ also requires that all occurrences of `ite` be compiled away by replacing $z = \text{ite}(p, x, y)$ with the equivalent $(p \supset z = x) \wedge (\bar{p} \supset z = y)$.

² It is shown in [10] that \mathcal{T}_{UF} and all fully parametric theories are flexible.

³ We view assignments as sets of literals; e.g. $\{\bar{p}, q, \bar{r}\}$ is the assignment that maps p, r to `false` and q to `true`.

Decide	$\frac{l \in L \quad l, \bar{l} \notin M}{M := M + \square + l}$
UnitPropag	$\frac{l \vee l_1 \vee \dots \vee l_k \in \Psi \quad \bar{l}_1, \dots, \bar{l}_k \in M \quad l, \bar{l} \notin M}{M := M + l}$
Conflict	$\frac{C = \text{no_cflct} \quad \bar{l}_1 \vee \dots \vee \bar{l}_k \in \Psi \quad l_1, \dots, l_k \in M}{C := \{l_1, \dots, l_k\}}$
Explain	$\frac{l \in C \quad l \vee \bar{l}_1 \vee \dots \vee \bar{l}_k \in \Psi \quad l_1, \dots, l_k \prec l}{C := C \cup \{l_1, \dots, l_k\} \setminus \{l\}}$
Learn	$\frac{C = \{l_1, \dots, l_k\} \quad \bar{l}_1 \vee \dots \vee \bar{l}_k \notin \Psi}{\Psi := \Psi \cup \{\bar{l}_1 \vee \dots \vee \bar{l}_k\}}$
BackJump	$\frac{C = \{l, l_1, \dots, l_k\} \quad \bar{l} \vee \bar{l}_1 \vee \dots \vee \bar{l}_k \in \Psi \quad \text{level } l > m \geq \text{level } l_i}{C := \text{no_cflct} \quad M := M^{[m]} + \bar{l}} \quad (i = 1, \dots, k)$

Fig. 2. Rules of DPLL. We prove in the appendix (Theorems its runs are finite) and has these basic correctness properties: if, initialized with the set of clauses Ψ_{init} , DPLL terminates in the state $\langle \Psi, M, C \rangle$, then : (a) $C = \text{no_cflct}$ or $C = \emptyset$; (b) If $C = \emptyset$ then Ψ_{init} is unsatisfiable; (c) If $C = \text{no_cflct}$, then M is a model for Ψ_{init} .

4 Combined Solver as a Transition System

We begin with DPLL—a transition system that models the operation of a DPLL-style SAT solver, including conflict analysis. Then we extend it to our main system NODPLL by adding rules for Nelson-Oppen cooperation of multiple theories.

4.1 DPLL with Conflict Analysis

The only parameter of the DPLL system is a finite set of propositional literals L , closed under negation. A DPLL *state* is a triple $\langle \Psi, M, C \rangle$, where: (1) Ψ is a set of clauses over L (2) M is a *checkpointed sequence* of literals in L , meaning that each element of M is either a literal or a special checkpoint symbol \square , (3) C is either a subset of L or a special symbol no_cflct .

The “input” to DPLL is an arbitrary set of clauses Ψ_{init} , modeled as an initial state in which Ψ is Ψ_{init} , M is empty, and C is no_cflct . The rules describing the state-to-state transitions are given in Figure 2. The rules have the guarded assignment form: above the line is the condition that enables the rule, below the line is the update to system variables Ψ, M, C .

The notation used in the rules is defined as follows. The negation of a literal l is \bar{l} . The relation $l \prec l'$ means that an occurrence of l precedes an occurrence

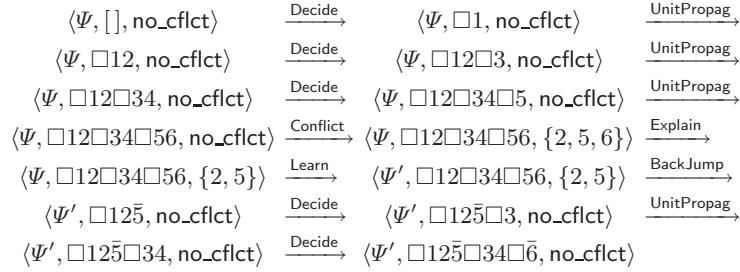


Fig. 3. An example run of DPLL. The initial set of clauses $\Psi = \{\bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee 6, \bar{2} \vee \bar{5} \vee \bar{6}\}$ is taken from [13]. The Learn move changes Ψ into $\Psi' = \Psi \cup \{\bar{2} \vee \bar{5}\}$. The BackJump move goes from decision level 3 to decision level 1. The final assignment satisfies Ψ .

of l' in M . Note that M can be written uniquely in the form $M = M^{(0)} + \square + M^{(1)} + \square + \dots + \square + M^{(d)}$, where $+$ denotes sequence concatenation and \square does not occur in any $M^{(i)}$. The superscripts indicate “decision levels”. A literal can occur at most once in M ; we write level $l = i$ if l occurs in $M^{(i)}$. Finally, $M^{[m]} = M^{(0)} + \square + \dots + \square + M^{(m)}$, the sequence up to decision level m .

4.2 Nelson-Oppen with DPLL

The parameters of our Nelson-Oppen-with-DPLL transition system NODPLL are pairwise disjoint theories $\mathcal{T}_1, \dots, \mathcal{T}_n$. Let $I = \{\mathbb{B}, \mathbb{E}, 1, \dots, n\}$. The state variables of NODPLL are the following:

- SHARED VARIABLE SETS V_i for $i \in I$. These sets are not necessarily disjoint; $V_{\mathbb{B}}$ contains only propositional variables, while the variables in other V_i may be of different types. For every $i \in I$ define the set L_i of INTERFACE LITERALS (paraphrasing [4]) to consist of variables in $V_i \cap V_{\mathbb{B}}$ and their negations, and also equalities of the form $x = y$, where $x, y \in V_i - V_{\mathbb{B}}$.
- LOCAL CONSTRAINTS Ψ_i for $i \in I$. $\Psi_{\mathbb{B}}$ is a set of clauses, $\Psi_{\mathbb{E}}$ is a set of $\mathcal{T}_{\mathbb{E}}$ -formulas, and Ψ_i for $i \in \{1, \dots, n\}$ is a set of (pure) \mathcal{T}_i -formulas and cardinality constraints. We require that $V_{\mathbb{B}}$ contains all variables in $\Psi_{\mathbb{B}}$, and that all variables that occur in both Ψ_i and Ψ_j (for distinct $i, j \neq \mathbb{B}, \mathbb{E}$) must occur in V_i, V_j , and $V_{\mathbb{E}}$.
- LOCAL STACKS (checkpointed sequences) M_i for $i \in I$. Any element of M_i is either \square or a LABELED LITERAL—a pair $\langle l, j \rangle$, where $l \in L_i$ and $j \in I$.
- The CONFLICT C —either a set of labeled literals or a special symbol `no_cflct`.

The “input” to NODPLL is a purified query $\Phi = \Phi_{\mathbb{B}} \cup \Phi_{\mathbb{E}} \cup \Phi_1 \cup \dots \cup \Phi_n$ as in Section 3, modeled as the INITIAL STATE s_{init}^{Φ} , in which, by definition, Ψ_i is $\Phi_i^{\text{pure}} \cup \Phi_i^{\text{card}}$ (assuming $\Psi_{\mathbb{B}}^{\text{card}} = \Psi_{\mathbb{E}}^{\text{card}} = \emptyset$), M_i is empty, C is `no_cflct`, $V_{\mathbb{B}} = \text{vars}(\Phi_{\mathbb{B}})$, $V_i = \bigcup_{j \neq i, \mathbb{E}} \text{vars}(\Phi_i) \cap \text{vars}(\Phi_j)$, and $V_{\mathbb{E}} = \bigcup_{i \neq \mathbb{B}} V_i$.

The transitions of NODPLL are defined by the rules in Figure 4. The following paragraphs explain the additional notation used in these rules.

Decide	$\frac{l \in L_{\mathbb{B}} \quad l, \bar{l} \notin M_{\mathbb{B}}}{M_{\mathbb{B}} := M_{\mathbb{B}} + \square + \langle l, \mathbb{B} \rangle \quad M_i := M_i + \square \quad (\text{all } i \neq \mathbb{B})}$
Infer _i	$\frac{l \in L_i \quad l, \bar{l} \notin M_i \quad \Psi_i, M_i \models_i l}{M_i := M_i + \langle l, i \rangle}$
LitDispatch _i ($i \neq \mathbb{B}$)	$\frac{l \in L_i \quad \langle l, j \rangle \in M_{\mathbb{B}} \quad l \notin M_i}{M_i := M_i + \langle l, j \rangle}$
EqDispatch _i ($i \neq \mathbb{E}, \mathbb{B}$)	$\frac{x, y \in V_i \quad \langle x = y, j \rangle \in M_{\mathbb{E}} \quad x = y \notin M_i}{M_i := M_i + \langle x = y, j \rangle}$
LitPropag _i ($i \neq \mathbb{B}$)	$\frac{l \in L_{\mathbb{B}} \quad \langle l, i \rangle \in M_i \quad l, \bar{l} \notin M_{\mathbb{B}}}{M_{\mathbb{B}} := M_{\mathbb{B}} + \langle l, i \rangle}$
EqPropag _i ($i \neq \mathbb{E}, \mathbb{B}$)	$\frac{x, y \in V_{\mathbb{E}} \quad \langle x = y, i \rangle \in M_i \quad x = y \notin M_{\mathbb{E}}}{M_{\mathbb{E}} := M_{\mathbb{E}} + \langle x = y, i \rangle}$
Conflict _i	$\frac{C = \text{no_cflct} \quad \langle l_1, i_1 \rangle, \dots, \langle l_k, i_k \rangle \in M_i \quad \Psi_i, l_1, \dots, l_k \models_i \text{false}}{C := \{ \langle l_1, i_1 \rangle, \dots, \langle l_k, i_k \rangle \}}$
Explain _i	$\frac{\langle l, i \rangle \in C \quad \langle l_1, i_1 \rangle, \dots, \langle l_k, i_k \rangle \prec_{M_i} \langle l, i \rangle \quad \Psi_i, l_1, \dots, l_k \models_i l}{C := C \cup \{ \langle l_1, i_1 \rangle, \dots, \langle l_k, i_k \rangle \} \setminus \{ \langle l, i \rangle \}}$
Learn	$\frac{C^b = \{ l_1, \dots, l_k \} \quad C^b \subseteq L_{\mathbb{B}} \quad \bar{l}_1 \vee \dots \vee \bar{l}_k \notin \Psi_{\mathbb{B}}}{\Psi_{\mathbb{B}} := \Psi_{\mathbb{B}} \cup \{ \bar{l}_1 \vee \dots \vee \bar{l}_k \}}$
BackJump	$\frac{C^b = \{ l, l_1, \dots, l_k \} \quad \bar{l} \vee \bar{l}_1 \vee \dots \vee \bar{l}_k \in \Psi_{\mathbb{B}} \quad \text{level } l > m \geq \text{level } l_i \quad (i = 1, \dots, k)}{C := \text{no_cflct} \quad M_{\mathbb{B}} := M_{\mathbb{B}}^{[m]} + \langle \bar{l}, \mathbb{B} \rangle \quad M_i := M_i^{[m]} \quad (\text{all } i \neq \mathbb{B})}$

Fig. 4. Rules of NODPLL.

Forget	$\frac{C = \text{no_cflct} \quad \phi \in \Psi_{\mathbb{B}} \quad \Psi_{\mathbb{B}} - \{ \phi \} \models \phi}{\Psi_{\mathbb{B}} := \Psi_{\mathbb{B}} - \{ \phi \}}$
Restart	$\frac{C = \text{no_cflct}}{M_i := M_i^{[0]} \quad (\text{all } i)}$
ThLearn _i	$\frac{l_1, \dots, l_k \in L_{\mathbb{B}} \quad \Psi_i \models_i l_1 \vee \dots \vee l_k}{\Psi_{\mathbb{B}} = \Psi_{\mathbb{B}} \cup \{ l_1 \vee \dots \vee l_k \}}$

Fig. 5. Additional rules for NODPLL.

We write C^b for the set of (unlabeled) literals occurring in C . We write $l \in M_i$ to mean that $\langle l, j \rangle \in M_i$ holds for some j . For $i \neq \mathbb{B}, \mathbb{E}$, the symbol \models_i denotes entailment modulo \mathcal{T}_i ; $\models_{\mathbb{B}}$ is propositional entailment, and $\models_{\mathbb{E}}$ is entailment modulo $\mathcal{T}_{\mathbb{E}q}$. Note that the variables of Φ_i^{pure} have the same names as variables of Φ_i , but possibly different types. We take the liberty to write $\Psi_i, l_1, \dots, l_k \models_i l$ in the rule Explain_i even though it is not correctly typed; we mean the entailment $\Psi_i, l'_1, \dots, l'_k \models_i l'$ where each primed literal is obtained by replacing the Φ_i -variables occurring in it with the corresponding (equally named) Φ_i^{pure} -variables. The same convention is used in the rules Infer_i and Conflict_i .

It is easy to see that *all local stacks have the same number of occurrences of \square* . Thus, each M_i can be written as $M_i = M_i^{(0)} + \square + M_i^{(1)} + \square + \dots + \square + M_i^{(d)}$, where d is the CURRENT DECISION LEVEL and \square does not occur in any $M_i^{(k)}$. This is an important invariant of NODPLL: *In any local stack, any literal can occur at most once* (Appendix, Section E). Knowing this, we can correctly define $\langle l, j \rangle \prec_{M_i} \langle l', j' \rangle$ (notation used in the rule Explain) to mean that both labeled literals occur in M_i and that the occurrence of $\langle l, j \rangle$ precedes the occurrence of $\langle l', j' \rangle$. The function level used in the BackJump rule is defined only for literals that occur in $M_{\mathbb{B}}$; we have $\text{level } l = k$ if l occurs in $M_{\mathbb{B}}^{(k)}$.

Example 2. If NODPLL is initialized with $\Phi_{\mathbb{B}}, \Phi_{\mathbb{E}}, \Phi_{\text{UF}}, \Phi_{\text{Int}}$ from Example 1, the first 13 steps could be (1) $\text{Infer}_{\mathbb{B}}$, (2) $\text{Infer}_{\mathbb{B}}$, (3) $\text{LitDispatch}_{\mathbb{E}}$, (4) $\text{LitDispatch}_{\text{Int}}$, (5) $\text{Infer}_{\mathbb{E}}$, (6) $\text{EqDispatch}_{\text{Int}}$, (7) $\text{Infer}_{\text{Int}}$, (8) $\text{EqPropag}_{\text{Int}}$, (9) $\text{EqDispatch}_{\text{UF}}$, (10) $\text{EqDispatch}_{\text{UF}}$, (11) Infer_{UF} , (12) $\text{EqPropag}_{\text{UF}}$, (13) $\text{EqDispatch}_{\text{Int}}$, with these rules modifying the local stacks as follows:

$$\begin{aligned}
M_{\mathbb{B}} &: \square \xrightarrow{(1)} \boxed{p} \xrightarrow{(2)} [p, \boxed{q}] \\
M_{\mathbb{E}} &: \square \xrightarrow{(3)} [p] \xrightarrow{(5)} [p, \boxed{y = x}] \xrightarrow{(8)} [p, y = x, z = x] \xrightarrow{(12)} [p, y = x, z = x, u = y] \\
M_{\text{UF}} &: \square \xrightarrow{(9)} [y = x] \xrightarrow{(10)} [y = x, z = x] \xrightarrow{(11)} [y = x, z = x, \boxed{u = y}] \\
M_{\text{Int}} &: \square \xrightarrow{(4)} [q] \xrightarrow{(6)} [q, y = x] \xrightarrow{(7)} [q, y = x, \boxed{z = x}] \xrightarrow{(13)} [q, y = x, z = x, u = y]
\end{aligned}$$

We omit the labels in labeled literals; the highlighted occurrence of each literal indicates its label in *all* stacks. The execution terminates in 6 more steps $\text{Conflict}_{\text{Int}}, \text{Explain}_{\text{UF}}, \text{Explain}_{\text{Int}}, \text{Explain}_{\mathbb{E}}, \text{Explain}_{\mathbb{B}}, \text{Explain}_{\mathbb{B}}$ that transform C as follows: $\text{no_cflct} \rightarrow \{q, y = u\} \rightarrow \{q, z = x\} \rightarrow \{q, y = x\} \rightarrow \{q, p\} \rightarrow \{q\} \rightarrow \emptyset$.

We note that restricting NODPLL so that its rules $\text{Infer}_{\mathbb{B}}, \text{Conflict}_{\mathbb{B}}, \text{Explain}_{\mathbb{B}}$ are exactly as in DPLL (where they have stronger guards) would not change any of the system's major properties; see Section D.2.

4.3 Correctness

A labeled literal may occur in more than one local stack. As local stacks grow and shrink during a run of NODPLL, the same labeled literal may become “created” (by rules Decide , Infer , or BackJump), multiply its presence in local stacks (the dispatch and propagation rules), then partially or entirely disappear from the stacks (rule BackJump), then become created again etc. The labels in labeled

literals are used to track this intricate dynamics that makes NODPLL significantly more complex than DPLL. Theorem 2 shows that the basic correctness still holds. The completeness issue needs further study; in Section F we prove it for the classical case when theories are convex and there are no cardinality constraints, and also when all equality literals are proxied (the “*MathSAT* approach”; see Section 4.4).

Theorem 2. *(a) Termination: Every run of NODPLL is finite and ends in a state where $C = \text{no_cflct}$ or $C = \emptyset$. (b) Soundness: If a final state with $C = \emptyset$ is reachable, then the input query is T -unsatisfiable.*

4.4 Comments

By design, NODPLL allows a great deal of freedom for implementation. We illustrate its flexibility and adequacy for expressing important design decisions on several examples, without elaboration.

Basic Extensions. The rules *Forget*, *Restart*, *ThLearn* in Figure 5 are used in practice to improve the boolean search. Reasonably restricting their use to preserve termination, they can be safely added to the basic NODPLL. This has been addressed in [13].

NODPLL keeps the variable and constraint sets V_i, Ψ_i constant, except for $\Psi_{\mathbb{B}}$, which grows with each *Learn* or *ThLearn* step and shrinks with *Forget* or *Restart*. However, it is useful to also have rules that modify V_i and/or Ψ_i . A simple rule that replaces Ψ_i with an equisatisfiable Ψ'_i that may even use fresh variables [6, 13] would allow us to generate $(x \notin u \wedge x \in v) \vee (x \in u \wedge x \notin v)$ when $u \neq v$ (in set theory) and $u = 3x - 1 \wedge v = 2x + 1$ when $2u + 3y = 1$ (in integer arithmetic), where in both cases x is fresh. Adding new variables (proxies of theory facts) to $V_{\mathbb{B}}$ is the essence of “splitting on demand” [3]; together with the above Ψ_i -modifying rule and *ThLearn*, it models Extended T-Learn of [3]. Termination is an issue again, but manageable, as shown in [3].

Equality Proxying and Propagation. If in Example 2 we had a proxy for $y = u$, say $r \Leftrightarrow y = u$, with this definitional form put in M_{Int} and M_{UF} , and with r in $V_{\mathbb{B}}$, then instead of propagating $y = u$ from M_{UF} to M_{Int} with *EqPropag* and *EqDispatch* in steps 12 and 13, we can equivalently propagate r with *LitPropag* and *LitDispatch*. Proxying every equality between shared variables like this up front would allow us to eliminate the rules *EqPropag*, *EqDispatch* altogether, as introduced by *MathSAT* [4]. *Yices* [9] goes a step further and curtails proliferation of proxies by introducing them only “on demand”. On the other hand, in *Simplify* [8] and in *CVC Lite* [2] propagation of equalities can occur directly, without creating propositional proxy variables.

Participating Solvers. The requirements on the deductive strength of individual solvers can be read off from Figure 4 and are exactly as specified by the *DPLL(T)* framework [13]. *Infer* needs deduction of new literals from those in

the local stack and **Explain** needs a subset (preferably small) of the local stack that suffices for that deduction. **Conflict** requires detection of inconsistency and an inconsistent (small again) subset of the local stack. How complete a solver needs to be with respect to **Infer** and **Conflict** depends on the approach taken with equality proxying (and on the convexity if of the theory) and is discussed in [13]. There is also an explicit requirement to deal with cardinality constraints, which sometimes can be delegated to the equality module and the SAT solver by introducing clauses saying that every variable of a specific finite type is equal to one of (representatives of) elements of that type [9]. This issue has been only recently raised [10] and is awaiting proper treatment.

5 Conclusion

Evidently, verification software should itself be based on clear and well-understood algorithmics, but in the SMT area, gaps between published algorithms and their actual implementations are common. Clarification efforts are needed and recent work on the *DPPL(T)* architecture [13] shows how such efforts pay off with superior implementations. Our involvement in the design of a comprehensive SMT solver prompted the question of what exactly is “*DPLL(T)* with Nelson-Oppen”, but the available answers were lacking in various ways. With the transition system *NODPLL* presented in this paper, we have identified an abstraction layer for describing SMT solvers that is fully tractable by formal analysis and comfortably close to implementation. It gives a precise setting in which one can see the features of existing systems, search for improvements, and recount them.

Acknowledgments We thank Jim Grundy, Murali Talupur and Cesare Tinelli for their comments, and Leonardo de Moura for explaining some *Yices* details.

This appendix is a work in progress. Please check later for more details, examples, and more polished arguments.

Appendix: Proofs and Comments

Contents

- A Type Instantiation and Satisfiability
- B A Purification Lemma
- C Correctness of DPLL
- D Comments on NODPLL
- E Termination and Soundness of NODPLL (Proof of Theorem 2)
- F Completeness of NODPLL

A Type Instantiation and Satisfiability

Assuming the notation of Section 3.1, we prove here the following precise formulation of the claim made at the end of that section.

Lemma 1. *Suppose \mathcal{T}_i is a flexible and permutationally invariant theory. An i -semipure query Φ is \mathcal{T} -satisfiable if and only if $\Phi^{\text{pure}} \cup \Phi^{\text{card}}$ is \mathcal{T}_i -satisfiable.*

For the additional unexplained terminology, we refer to [10]. Regarding the flexibility condition, it is shown there that virtually every datatype theory of interest satisfies it. Also, all interesting theories are PERMUTATIONALLY INVARIANT, by which we mean for every “type environment isomorphism” $R: \iota_1 \leftrightarrow \iota_2$ (every component $R(\alpha): \iota_1(\alpha) \leftrightarrow \iota_2(\alpha)$ is a bijection) and any two environments $\langle \iota_1, \rho_1 \rangle$ and $\langle \iota_2, \rho_2 \rangle$ for a term t such that

$$\langle \rho_1(v^\tau), \rho_2(v^\tau) \rangle \in \llbracket \tau \rrbracket^\# R \quad \text{for all } v^\tau \in \text{vars}(t)$$

we must have

$$\langle \llbracket t \rrbracket \langle \iota_1, \rho_1 \rangle, \llbracket t \rrbracket \langle \iota_2, \rho_2 \rangle \rangle \in \llbracket \sigma \rrbracket^\# R$$

Thus, permutational invariance is parametricity restricted to bijective relations; it follows from full parametricity and from Reynolds parametricity as well. See Theorems 2 and 3 of [10].

Suppose θ is a type instantiation. Recall that $\theta(t)$ is the term obtained by replacing every variable x^σ in t with $x^{\theta(\sigma)}$. Thus,

$$\text{vars}(\theta(t)) = \{x_1^{\theta(\sigma_1)}, \dots, x_n^{\theta(\sigma_n)}\}, \text{ where } \text{vars}(t) = \{x_1^{\sigma_1}, \dots, x_n^{\sigma_n}\} \quad (2)$$

Given an environment $\langle \iota, \rho \rangle$ for $\theta(t)$, there is a uniquely defined environment $\langle \iota \cdot \theta, \rho \cdot \theta \rangle$ for t . The definitions are:

$$(\iota \cdot \theta)(\alpha) = \llbracket \theta(\alpha) \rrbracket \iota \quad (3)$$

$$(\rho \cdot \theta)(x^\sigma) = \rho(x^{\theta(\sigma)}) \quad (4)$$

These equalities generalize to arbitrary types and terms as follows:

$$\llbracket \sigma \rrbracket \langle \iota \cdot \theta \rangle = \llbracket \theta(\sigma) \rrbracket \iota \quad (5)$$

$$\llbracket t \rrbracket \langle \iota \cdot \theta, \rho \cdot \theta \rangle = \llbracket \theta(t) \rrbracket \langle \iota, \rho \rangle \quad (6)$$

When $\sigma = \alpha$, (5) is exactly (3)—the base case of a simple proof of (5) by induction on the structure of σ . Note that (5) is needed to ensure that (4) is a correct definition. Note also that when $t = x^\sigma$, (6) is exactly (4). Again, with this as a base case, the induction on the structure of t proves (6).

Given a type instantiation θ , we define θ^{card} to be the set of all cardinality constraints $\alpha \doteq n$, where $\alpha \in \text{dom}(\theta)$ and $\theta(\alpha)$ is a finite ground type with n elements.

We will derive Lemma 1 from the following more general fact.

Lemma 2. *Let $\mathcal{T} = \mathcal{T}' + \mathcal{T}''$ and let \mathcal{T}' be flexible and permutation-invariant. Let ϕ be a \mathcal{T}' -formula and θ a type instantiation that maps type variables to \mathcal{T} -terms. Then*

$$\theta(\phi) \text{ is } \mathcal{T}\text{-equisatisfiable} \iff \phi, \theta^{\text{card}} \text{ is } \mathcal{T}'\text{-satisfiable}$$

Proof. If $\langle \iota, \rho \rangle$ is a \mathcal{T} -model for $\theta(\phi)$, then by (6), $\langle \iota \cdot \theta, \rho \cdot \theta \rangle$ is a \mathcal{T} -model for ϕ and so a \mathcal{T}' -model for ϕ . (For ϕ , \mathcal{T} -satisfiability and \mathcal{T}' -satisfiability are, of course, the same.) Moreover, if $\theta(\alpha)$ is a ground type of cardinality n , then by (3), $\langle \iota \cdot \theta, \rho \cdot \theta \rangle \models \alpha \doteq n$. This proves the easy half of the lemma.

Suppose now $\langle \iota, \rho \rangle$ is a model for ϕ and the cardinality constraints θ^{card} . Our goal is to create a model $\langle \iota', \rho' \rangle$ for $\theta(\phi)$. To define the type environment ι' , simply pick an arbitrary countably infinite set \mathbb{I} and set $\iota'(\beta) = \mathbb{I}$ for every variable β in $\cup_{\alpha \in \text{dom}(\theta)} \text{tyvars}(\theta(\alpha))$. This implies that $\llbracket \sigma \rrbracket \iota'$ is finite if and only if σ is a finite ground type.⁴

Let $\iota_1 = \iota' \cdot \theta$. Observe that for any $\alpha \in \text{dom}(\iota)$, if $\theta(\alpha)$ is a finite ground type of cardinality n , then both sets $\iota(\alpha)$ and $\iota_1(\alpha)$ have cardinality n . Indeed, for $\iota(\alpha)$ this is true because of our assumption that ι satisfies the constraints θ^{card} , and for ι_1 it is true by definition; see (3). If $\theta(\alpha)$ is *not* a finite ground type, then (as marked above) $\iota_1(\alpha)$ is an infinite set, say of cardinality κ . Depending on whether $|\iota(\alpha)| < \kappa$ or not, we can use the up- or down-flexibility of \mathcal{T}' to change the model $\langle \iota, \rho \rangle$ of ϕ so that the cardinality of $\iota(\alpha)$ becomes κ , without affecting $\iota(\alpha')$ for any $\alpha' \neq \alpha$. Adjusting thus the cardinality of each $\iota(\alpha)$ if necessary, we get into a situation where $|\iota(\alpha)| = |\iota_1(\alpha)|$ holds for every $\alpha \in \text{dom}(\iota)$. Assume then that we are indeed in that situation.

Choose arbitrary bijections $f_\alpha : \iota(\alpha) \rightarrow \iota_1(\alpha)$ for $\alpha \in \text{dom}(\iota)$ and let $f_\sigma : \llbracket \sigma \rrbracket \iota \rightarrow \llbracket \sigma \rrbracket \iota_1$ be the induced bijections, defined for every σ with $\text{tyvars}(\sigma) \supseteq \text{dom}(\iota)$. (Precisely, $f_\sigma = \llbracket \sigma \rrbracket^\sharp f$, where $f : \iota \leftrightarrow \iota_1$ is the collection of bijections f_α .) Define

⁴ Here we assume that if at least one of the arguments of a type constructor is an infinite set, then the result is infinite as well. For type constructors of interest, this is true without exception.

ρ_1 by $\rho_1(x^\sigma) = f_\sigma(\rho(x^\sigma))$. It follows by the permutation-invariance of \mathcal{T}' that $\langle \iota_1, \rho_1 \rangle \models \phi$.

Now it just remains to define ρ' by $\rho'(x^{\theta(\sigma)}) = \rho_1(x^\sigma)$. For correctness of this definition, notice that every variable in $\text{vars}(\theta(\phi))$ is of the form $x^{\theta(\sigma)}$, where $x^\sigma \in \text{vars}(\phi)$ (equation (2)). Thus, we have $\iota' \cdot \theta = \iota_1$ and $\rho' \cdot \theta = \rho_1$. Equation (6) combined with $\langle \iota_1, \rho_1 \rangle \models \phi$ immediately implies $\langle \iota', \rho' \rangle \models \theta(\phi)$, which was to be proved. \square

Proof of Lemma 1. Just take \mathcal{T}' to be \mathcal{T}_i and \mathcal{T}'' to be the sum of all \mathcal{T}_j where $j \neq i$. Let θ be the type instantiation such that $\theta(\Phi^{\text{pure}}) = \Phi$. Then $\theta^{\text{card}} = \Phi^{\text{card}}$, and just apply Lemma 2. \square

B A Purification Lemma

Lemma 3. *Let the notation be as in Section 3.2. Assume that Bool can occur in the arity of a non-logical constant⁵ only as the result type. Then for every query Φ there exists a query Φ' such that every formula in Φ' is either a propositional clause or has one of the following DEFINITIONAL FORMS:*

$$(A) \ p \Leftrightarrow (x = y) \quad (B) \ p \Leftrightarrow \phi \quad (C) \ x = t$$

where (i) p, x, y are variables, ϕ is a strictly semipure literal, and t is a strictly semipure term; (ii) every variable occurs as the left-hand side in at most one definitional form; (iii) no (propositional) variable that occurs in propositional clauses of Φ' can occur in the right-hand side of any definitional form.

Proof. We give an (unoptimized) algorithm that transforms Φ into Φ' which has the required properties. Each step of the algorithm transforms a pair $\langle \Phi_0, \Phi_1 \rangle$ of queries as described below. At the beginning, set $\Phi_0 = \Phi$ and $\Phi_1 = \emptyset$. Each step will add to Φ_1 some definitional forms and bring Φ_0 closer to a propositional query (by either decreasing the number of occurrences of non-boolean constants in Φ_0 , or by decreasing the size of a non-propositional formula in Φ_0). At the end, Φ_0 will be a propositional query, and we will have $\Phi' = \Phi_0 \cup \Phi_1$. At each step, the union $\Phi_0 \cup \Phi_1$ will be kept in the same equisatisfiability class.

We can assume that the input Φ does not have occurrences of `ite`. If it does, then in a preprocessing step, we can get rid of them by replacing each occurrence of a term `ite`(ϕ, t_1, t_2) in Φ with a fresh variable z and adding the formulas $(\phi \supset z = t_1) \wedge (\bar{\phi} \supset z = t_2)$ to Φ_0 . This is clearly an equisatisfiable transformation.

First, introduce new variables \mathbb{k} and \mathbb{f} , replace all occurrences of `true` and `false` in Φ_0 with \mathbb{k} and \mathbb{f} , and add $\mathbb{k} = \text{true}$ and $\mathbb{f} = \text{false}$ to Φ_1 . Then repeat the following steps for as long as possible.

- (1) Replace in Φ_0 a subterm of the form $k x_1 \dots x_m$, where k is a non-logical symbol and the x_i are variables, with a fresh variable z , and add either $z \Leftrightarrow k x_1 \dots x_m$ or $z = k x_1 \dots x_m$ to Φ_1 (depending on whether k is a predicate or not).

⁵ Recall that the logical constants are those in Σ_{Eq} .

- (2) Replace in Φ_0 a subterm of the form $x = y$, where x and y are variables, with a fresh variable z , and add $z \Leftrightarrow (x = y)$ to Φ_1 .
- (3) Replace an occurrence of a (non-variable) propositional formula ϕ in Φ_0 with a fresh variable z and add $z \Leftrightarrow \phi$ to Φ_0 .

It is easy to see that we can apply these steps only finitely many times; when none of these steps is applicable, then Φ_0 is a propositional query, and Φ_1 is a set of definitional forms such that $\Phi_0 \cup \Phi_1$ satisfies conditions (i) and (ii). To make (iii) true too, eliminate first all occurrences of \mathbb{t} and \mathbb{f} from Φ_0 , which is clearly possible. (If \mathbb{f} is a formula in Φ_0 , then $\Phi_0 \cup \Phi_1$ is unsatisfiable, so we can collapse $\Phi_0 \cup \Phi_1$ to $\{\mathbb{f}, \mathbb{f} = \text{false}\}$.) Then for each propositional variable p that occurs both in Φ_0 and in the right-hand side of definitional forms in Φ_1 , introduce fresh variables p', p'' , add $p \Leftrightarrow p''$ to Φ_0 , add the definitional form $p'' = (p' = \mathbb{t})$ to Φ_1 , and replace all occurrences of p with p' in the right-hand sides of definitional forms in Φ_1 . \square

C Correctness of DPLL

Theorems 3 and 4 below prove the properties mentioned in the caption of Figure 2. Even though these theorems can be derived from Theorem 2, we prove them here because they are perhaps of independent interest, and also because their proofs may serve as an introduction to the more involved proof of Theorem 2.

Theorem 3 (Termination). *Every run of DPLL is finite.* \square

Proof. If M and M' are checkpointed sequences, define $M \preceq M'$ to mean that either $M = M'$, or for some m , $M^{[m]}$ is a proper prefix of $M'^{[m]}$. It is easy to prove that \preceq is a partial order.

Observe that the rules `Decide`, `UnitPropag` and `BackJump` increase the M -component of the DPLL state, while the other three rules leave M unchanged. Observe also that the set of all possible values for M is finite, as a consequence of this easily checked invariant of DPLL:

$$\text{If } l \text{ occurs in } M, \text{ then it occurs only once and } \bar{l} \text{ does not occur at all.} \quad (7)$$

It follows now that in every run

$$s_{\text{init}} \rightarrow s_1 \rightarrow s_2 \rightarrow \dots \quad (8)$$

of DPLL, only finitely many steps are based on `Decide`, `UnitPropag` and `BackJump`. Since there must be a `BackJump` between any two occurrences of `Conflict` (to restore $C = \text{no_cflct}$), the number of occurrences of `Conflict` in (8) is also finite. The same is true about `Learn` because there are only finitely many clauses to add to Ψ . Thus, the only possibility for (8) to be infinite is that from some point on, all steps are based on the rule `Explain`. To see that every sequence of `Explain`-based steps must be finite, observe first an easy invariant of DPLL:

$$\text{Every literal in } C \text{ occurs in } M. \quad (9)$$

Notice then that, leaving M intact, **Explain** replaces a literal l of C with a set of literals that precede l in M . Thus, with each application of **Explain**, the set C gets smaller in the multisided ordering that the ordering of M induces on subsets of M .⁶ \square

Theorem 4 (Correctness). *Let $s_\diamond = \langle \Psi_\diamond, M_\diamond, C_\diamond \rangle$ be a state obtained by running DPLL initialized with the set of clauses Ψ_{init} . (The state s_\diamond is “final” in the sense that no DPLL rule applies to it.) Then*

- (a) $C_\diamond = \text{no_cflct}$ or $C_\diamond = \emptyset$;
- (b) If $C_\diamond = \emptyset$ then Ψ_{init} is unsatisfiable;
- (c) If $C_\diamond = \text{no_cflct}$, then M_\diamond is a model for Ψ_{init} . \square

Proof. (a) Assuming the contrary of the property (a), suppose C_\diamond is a non-empty set and let l be an arbitrary element of it. By (9), l occurs in M_\diamond . Considering a particular run $s_{\text{init}} \rightarrow s_1 \rightarrow s_2 \dots \rightarrow s_\diamond$, let s_n be the last state in this run such that $s_n.M$ does not contain l . Note that the transition $s_n \rightarrow s_{n+1}$ must be based on **Decide**, **UnitPropag**, or **BackJump**, since the other three rules do not update M . Note that $s_{n+1}.M = s_n.M + l$ and also that that

$$s_{n+1}.M \text{ is a prefix of } s_i.M \text{ for all } i > n. \quad (10)$$

Indeed, if (10) is not true, then in the execution $s_{n+1} \rightarrow \dots \rightarrow s_\diamond$ there must be a **BackJump** to the level smaller than the level of l , and it must remove l from M , contradicting the assumption that l occurs in $s_i.M$ for all $i > n$.

If the transition $s_n \rightarrow s_{n+1}$ is a **UnitPropag** or **BackJump**, then **Explain** is applicable to s_{n+1} and so by (10), **Explain** is applicable to s_\diamond as well, contradicting the finality of s_\diamond . Thus, our transition must be based on the rule **Decide**, and so l is a *decision literal* (a literal immediately following an occurrence of \square) in $s_\diamond.M$. Since l was by assumption an arbitrary element of C_\diamond , it follows that *all* elements of C_\diamond are decision literals. As a consequence, the levels of literals in C_\diamond are all distinct. Now, depending on whether the clause $\bigvee_{l \in C_\diamond} \bar{l}$ is in Ψ or not, either **BackJump** or **Learn** applies to s_\diamond . In both cases we run into contradiction with the finality of s_\diamond .

(b) We prove by simultaneous induction that the following two properties are invariants of DPLL. Note that (b) is a special case of (12) when $k = 0$.

$$\Psi \text{ is equivalent with } \Psi_{\text{init}} \quad (11)$$

$$\text{If } C = \{l_1, \dots, l_k\}, \text{ then } \Psi_{\text{init}} \models \bar{l}_1 \vee \dots \vee \bar{l}_k \quad (12)$$

Both properties are trivially true for initial states. Assuming $s \rightarrow s'$ is a transition based on a rule R of NODPLL, and s satisfies (11) and (12), we proceed to prove that s' satisfies these conditions too.

⁶ “ $M \sqsupset_{\text{mul}} N$ holds iff you can get from M to N by carrying out the following procedure one or more times: remove an element x and add a finite number of elements, all of which are smaller than x .” [1]

If R is not **Learn**, then $s'.\Psi = s.\Psi$ and so s' satisfies (11) by induction hypothesis. If R is **Learn**, the proof that s' satisfies (11) is a simple application of the induction hypotheses (11) and (12). Thus, s' satisfies (11) in all cases.

Now we prove that s' satisfies (12). The only non-trivial cases are when R is **Conflict_i** or **Explain_i**, since in all other cases we have $s'.C = s.C$. When R is **Conflict_i**, from the guard $\bar{l}_1 \vee \dots \vee \bar{l}_k \in s.\Psi$ and (11), we obtain the desired relation $\Psi_{\text{init}} \models \bar{l}_1 \vee \dots \vee \bar{l}_k$ immediately.

Suppose R is **Explain_i** and let γ be the clause consisting of the inverses of literals of $s.C - \{l\}$. By induction hypothesis, we have $\Psi_{\text{init}} \models \gamma \vee \bar{l}$. Also, the guard of **Explain** and (11) imply $\Psi_{\text{init}} \models \bar{l}_1 \vee \dots \vee \bar{l}_k \vee l$. The required relation $\Psi_{\text{init}} \models \gamma \vee \bar{l}_1 \vee \dots \vee \bar{l}_k$ follows immediately.

(c) From (9) and non-applicability of **Decide**, it follows that M_\diamond is an assignment: for every $l \in L$, it contains either l or \bar{l} . By (11), we only need to prove $M_\diamond \models \gamma$ for every clause $\gamma = l_1 \vee \dots \vee l_k$ in Ψ_\diamond . But if this were not true, we would have $\bar{l}_i \in M_\diamond$ for $i = 1, \dots, k$ and so **Conflict** would be applicable to s_\diamond , contradicting the finality of s_\diamond . \square

D Comments on DPLL and NODPLL

D.1 A strategy for DPLL

The guard of **Learn** is expensive and SAT solvers don't implement it. They don't need to. We can prove that for particular strategies of DPLL (and NODPLL) this guard is redundant.

[— Explain “MiniSAT” as a strategy for DPLL. —]

D.2 Strengthening the propositional rules of NODPLL

Clearly, DPLL is a subsystem of NODPLL, but its rules **UnitPropag**, **Conflict** and **Explain** are more permissive than the corresponding rules **Infer_B**, **Conflict_B** and **Explain_B** of NODPLL. If we change these three rules of NODPLL as follows, then the correspondence would be exact.

$$\begin{array}{l}
\text{Infer}_{\mathbb{B}} \quad \frac{l \in L_{\mathbb{B}} \quad l, \bar{l} \notin M_{\mathbb{B}} \quad l \vee l_1 \vee \dots \vee l_k \in \Psi \quad \bar{l}_1, \dots, \bar{l}_k \in M_{\mathbb{B}}}{M_{\mathbb{B}} := M_{\mathbb{B}} + \langle l, \mathbb{B} \rangle} \\
\text{Conflict}_{\mathbb{B}} \quad \frac{C = \text{no_cflct} \quad \langle l_1, i_1 \rangle, \dots, \langle l_k, i_k \rangle \in M_{\mathbb{B}} \quad \bar{l}_1 \vee \dots \vee \bar{l}_k \in \Psi_{\mathbb{B}}}{C := \{ \langle l_1, i_1 \rangle, \dots, \langle l_k, i_k \rangle \}} \\
\text{Explain}_{\mathbb{B}} \quad \frac{\langle l, \mathbb{B} \rangle \in C \quad \langle l_1, i_1 \rangle, \dots, \langle l_k, i_k \rangle \prec_{M_{\mathbb{B}}} \langle l, \mathbb{B} \rangle \quad l \vee \bar{l}_1 \vee \dots \vee \bar{l}_k \in \Psi_{\mathbb{B}}}{C := C \cup \{ \langle l_1, i_1 \rangle, \dots, \langle l_k, i_k \rangle \} \setminus \{ \langle l, \mathbb{B} \rangle \}}
\end{array}$$

This change would not affect any major properties of NODPLL. In particular, the proof of Theorem 2 given below would apply verbatim to the modified system.

D.3 Why do we need labeled literals in NODPLL?

Every literal that gets added to any of the local stacks is a logical consequence of the decision literals and the initialization query. When a conflict is reached, we need to identify a subset of decision literals that is sufficient for the conflict. The “explanation” mechanism serves this purpose; starting with a conflicting set of literals, it picks a non-decision literal from the set, detects the inference step that created it, and replaces the literal with a subset of the local stack that was used by the inference step. The result is a new conflicting set of literals that is in a precise sense “older”, so that repeating the “explaining a literal” process will terminate (with a conflicting set of decision literals).

Detecting the inference step that created a particular literal is the main purpose of labels: the label in a labeled literal simply signifies the theory responsible for its derivation. Without labels, the explanation process is too ambiguous and potentially circular. We give an example to illustrate.

Consider these five steps in a possible execution of NODPLL, where we indicate only the changes of local stacks $M_{\mathbb{B}}, M_1, M_2$, the other parts of the NODPLL state being unaffected. Suppose that, as shown, we use non-labeled literals in the local stacks.

$$\begin{array}{c} \begin{bmatrix} M_{\mathbb{B}} \\ M_1 \\ M_2 \end{bmatrix} \xrightarrow{\text{Infer}_1} \begin{bmatrix} M_{\mathbb{B}} \\ M_1 + l \\ M_2 \end{bmatrix} \xrightarrow{\text{LitPropag}_1} \begin{bmatrix} M_{\mathbb{B}} + l \\ M_1 + l \\ M_2 \end{bmatrix} \xrightarrow{\text{Infer}_{\mathbb{B}}} \\ \begin{bmatrix} M_{\mathbb{B}} + l + l' \\ M_1 + l \\ M_2 \end{bmatrix} \xrightarrow{\text{LitDispatch}_2} \begin{bmatrix} M_{\mathbb{B}} + l + l' \\ M_1 + l \\ M_2 + l' \end{bmatrix} \xrightarrow{\text{Infer}_2} \begin{bmatrix} M_{\mathbb{B}} + l + l' \\ M_1 + l \\ M_2 + l' + l \end{bmatrix} \end{array}$$

We can assume that $\bar{l} \vee l' \in \Psi_{\mathbb{B}}$ and $\Psi_2, l' \models_2 l$ —the facts responsible for the $\text{Infer}_{\mathbb{B}}$ and Infer_2 steps above. Suppose that sometime in the future of this execution a conflict arises, with the conflict set that contains l but not l' . Say, $C = \{l\} \cup D$. Then Explain_2 applies and changes C to $\{l'\} \cup D$. But then $\text{Explain}_{\mathbb{B}}$ applies and changes C back to $\{l\} \cup D$. So the system can repeat explaining l and l' with each other forever.

Consider now the same five steps, but with labeled literals:

$$\begin{array}{c} \begin{bmatrix} M_{\mathbb{B}} \\ M_1 \\ M_2 \end{bmatrix} \xrightarrow{\text{Infer}_1} \begin{bmatrix} M_{\mathbb{B}} \\ M_1 + \langle l, 1 \rangle \\ M_2 \end{bmatrix} \xrightarrow{\text{LitPropag}_1} \\ \begin{bmatrix} M_{\mathbb{B}} + \langle l, 1 \rangle \\ M_1 + \langle l, 1 \rangle \\ M_2 \end{bmatrix} \xrightarrow{\text{Infer}_{\mathbb{B}}} \begin{bmatrix} M_{\mathbb{B}} + \langle l, 1 \rangle + \langle l', \mathbb{B} \rangle \\ M_1 + \langle l, 1 \rangle \\ M_2 \end{bmatrix} \xrightarrow{\text{LitDispatch}_2} \\ \begin{bmatrix} M_{\mathbb{B}} + \langle l, 1 \rangle + \langle l', \mathbb{B} \rangle \\ M_1 + \langle l, 1 \rangle \\ M_2 + \langle l', \mathbb{B} \rangle \end{bmatrix} \xrightarrow{\text{Infer}_2} \begin{bmatrix} M_{\mathbb{B}} + \langle l, 1 \rangle + \langle l', \mathbb{B} \rangle \\ M_1 + \langle l, 1 \rangle \\ M_2 + \langle l', \mathbb{B} \rangle + \langle l, 2 \rangle \end{bmatrix} \end{array}$$

The conflict set C is now $\{\langle l, 2 \rangle\} \cup D$. It becomes $\{\langle l', \mathbb{B} \rangle\} \cup D$ after Explain_2 , and then becomes $\{\langle l, 1 \rangle\} \cup D$ after $\text{Explain}_{\mathbb{B}}$. At this point, we cannot explain $\langle l, 1 \rangle$ with $\langle l', \mathbb{B} \rangle$, and this prevents the circularity that was possible when labels were not used. Instead, rule Explain_1 will explain $\langle l, 1 \rangle$, with whatever literals were used in the Infer_1 step above.

D.4 Types of shared variables in local constraints

Consider again Example 1. Purification of the input $(\mathcal{T}_{\text{Int}} + \mathcal{T}_{\text{UF}})$ -query $\Phi = \{f(x^{\text{Int}}) = x^{\text{Int}} \wedge f(2x^{\text{Int}} - f(x^{\text{Int}})) > x^{\text{Int}}\}$ produces

$$\begin{aligned}\Phi_{\mathbb{B}} &= \{p^{\text{Bool}}, q^{\text{Bool}}\} \\ \Phi_{\mathbb{E}} &= \{p^{\text{Bool}} \Leftrightarrow (y^{\text{Int}} = x^{\text{Int}})\} \\ \Phi_{\text{UF}} &= \{y^{\text{Int}} = f(x^{\text{Int}}), u^{\text{Int}} = f(z^{\text{Int}})\} \\ \Phi_{\text{Int}} &= \{z^{\text{Int}} = 2x^{\text{Int}} - y^{\text{Int}}, q^{\text{Bool}} \Leftrightarrow u^{\text{Int}} > x^{\text{Int}}\}\end{aligned}$$

These semipure queries then generate pure queries for initialization of NODPLL:

$$\begin{aligned}\Psi_{\mathbb{B}} &= \{p^{\text{Bool}}, q^{\text{Bool}}\} \\ \Psi_{\mathbb{E}} &= \{p^{\text{Bool}} \Leftrightarrow (y^{\alpha} = x^{\alpha})\} \\ \Psi_{\text{UF}} &= \{y^{\beta} = f(x^{\beta}), u^{\beta} = f(z^{\beta})\} \\ \Psi_{\text{Int}} &= \{z^{\text{Int}} = 2x^{\text{Int}} - y^{\text{Int}}, q^{\text{Bool}} \Leftrightarrow u^{\text{Int}} > x^{\text{Int}}\}\end{aligned}$$

Thus, a variable of Φ (say, x^{Int}) has different “identities” in the pure queries Ψ_i (x^{α} in $\Phi_{\mathbb{E}}$, x^{β} in Ψ_{UF} , and x^{Int} in Ψ_{Int}); they all have the same name, but different types.

Suppose x^{σ} occurs in Φ , and $\sigma = F \sigma_1 \dots \sigma_k$, where F is a type constructor from \mathcal{T}_i . Then if Ψ_i contains a variable named x , its type will be of the form $F \sigma'_1 \dots \sigma'_k$. Also, if $j \neq i$ and Ψ_j contains a variable named x , then the type of x in Ψ_j is a type variable. See [10] for more details.

We formulated NODPLL so that the variables occurring in all local stacks are those of Φ . This requires correct interpretation of the guards $\Psi_i, M_i \models_i l$, $\Psi_i, l_1, \dots, l_k \models_i \text{false}$, $\Psi_i, l_1, \dots, l_k \models_i l$ of Infer_i , Conflict_i , Explain_i respectively. These guards are not well-typed as written; to avoid cluttering the notation, we tacitly assume (as indicated in Section 4.2) that the variables in M_i and the literals l, l_1, \dots, l_k here should be the Ψ_i -namesakes of the variables of Φ .

D.5 Propagation of disequalities

[— Comment a little. —]

E Termination and Soundness of NODPLL (Proof of Theorem 2)

We can assume that the input formula Φ is fixed, and will write s_{init} for s_{init}^{Φ} . The transition relation of NODPLL will be denoted by \rightarrow . For any state s , the union of local constraints $s.\Psi_{\mathbb{B}}, s.\Psi_{\mathbb{E}}, s.\Psi_1, \dots, s.\Psi_n$ will be denoted $s.\Psi$.

Let us start with a set of invariants of NODPLL:

$$\text{All local stacks contain the same number of occurrences of } \square. \quad (13)$$

$$\text{In any local stack, any literal can occur at most once.} \quad (14)$$

$$\text{If } l \text{ occurs in } M_{\mathbb{B}}, \text{ then } \bar{l} \text{ does not occur in } M_{\mathbb{B}}. \quad (15)$$

All these properties are obviously true in the initial state. Directly examining all rules, we can easily check that if $s \rightarrow s'$ and s satisfies the properties, then so does s' .

Consider an execution sequence (finite or infinite)

$$\pi : s_{\text{init}} = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots \quad (16)$$

For each state s_k in π we will collect the local stacks $s_k.M_i$ ($i = \mathbb{B}, \mathbb{E}, 1, \dots, n$) into the GLOBAL STACK $s_k.O$ that interleaves them all. The elements of that stack will be triples $\langle l, j, i \rangle$, denoting the occurrence of $\langle l, j \rangle$ in M_i . The global stacks $s_k.O$ are defined inductively as follows. First, $s_{\text{init}}.O$ is the empty sequence. Then, if the transition $s_k \rightarrow s_{k+1}$ is based on the rule R , let $s_{k+1}.O = s_k.O$ if R is Learn, Conflict _{i} or Explain _{i} ; if R is BackJump, let $s_{k+1}.O = (s_k.O)^{[m]} + \langle \bar{l}, \mathbb{B}, \mathbb{B} \rangle$; finally, if R is one of the remaining six rules, let $s_{k+1}.O = s_k.O + U$, where U is given in the following table:

$$\begin{array}{l} R : \quad \text{Decide} \quad \text{Infer}_i \quad \text{LitDispatch}_i \quad \text{EqDispatch}_i \quad \text{LitPropag}_i \quad \text{EqPropag}_i \\ U : \quad \square + \langle p^\epsilon, \mathbb{B}, \mathbb{B} \rangle \quad \langle l, i, i \rangle \quad \langle p^\epsilon, j, i \rangle \quad \langle x = y, j, i \rangle \quad \langle p^\epsilon, i, \mathbb{B} \rangle \quad \langle x = y, i, \mathbb{E} \rangle \end{array}$$

It follows immediately from the definition that $\langle l, i, j \rangle$ occurs in $s.O$ if and only if $\langle l, i \rangle$ occurs in $s.M_j$. The ordering of occurrence triples in $s.O$ will be denoted \sqsubseteq .

Lemma 4. *The following properties hold for all states in π .*

- (a) *If $\langle l, j \rangle \prec_{M_i} \langle l', j' \rangle$ then $\langle l, j, i \rangle \sqsubset \langle l', j', i \rangle$.*
- (b) *If $\langle l, i \rangle$ occurs in M_j , then it also occurs in M_i and we have $\langle l, i, i \rangle \sqsubseteq \langle l, i, j \rangle$.*

Proof. Straightforward induction. \square

Corollary 1. *If the rule LitDispatch _{i} or EqDispatch _{i} applies to a reachable state, then $j \neq i$. (The number j here is from the rule as stated in Figure 4.)*

Proof. Let s be a state in which LitDispatch _{i} applies. (We omit the discussion of the entirely analogous EqDispatch case.) Using an execution sequence π that contains s , we have $\langle p^\epsilon, j, i \rangle \in s.O$ and so, by part (b) of Lemma 4, $\langle p^\epsilon, j, j \rangle \in s.O$. In particular $p^\epsilon \in M_j$, which together with the guard $p^\epsilon \notin M_i$ implies $i \neq j$. \square

For each state s , define

$$s.C^* = \begin{cases} \text{no_cflct} & \text{if } s.C = \text{no_cflct} \\ \{ \langle l, i, i \rangle \mid \langle l, i \rangle \in s.C \} & \text{otherwise} \end{cases}$$

Lemma 5. (a) For every reachable state s with $s.C \neq \text{no_cflct}$, all elements of $s.C^*$ occur in $s.O$.

(b) If s is a reachable state and $s \rightarrow s'$ is a transition based on the rule Explain_i , then $s'.C^* \sqsubseteq_{\text{mul}} s.C^*$, where \sqsubseteq_{mul} is the multiset ordering⁷ induced by the relation \sqsubseteq on the set of triples occurring in $s'.O = s.O$.

Proof. First we prove (a) by induction. Let s, s' be two consecutive states in any execution sequence π and let the transition $s \rightarrow s'$ be based on a rule R . If R is any of the first six rules in Figure 4, or if R is Learn , then $s'.C^* = s.C^*$ and $s'.O$ contains $s.O$. If R is BackJump , then $s'.C \neq \text{no_cflct}$. In all these cases, there is nothing to check. Finally, if R is either Conflict_i or Explain_i , then $s'.O = s.O$, and $s'.C - s.C$ consists of pairs $\langle l_\nu, i_\nu \rangle$ that occur in $s.M_i$. Thus, $\langle l_\nu, i_\nu, i \rangle$ occurs in $s.O$, and by Lemma 4(a), the new element $\langle l_\nu, i_\nu, i_\nu \rangle$ of $s.C^*$ occurs in $s.O$ too.

For part (b), we have

$$s'.C = s.C - \{\langle l, i \rangle\} + \{\langle l_1, i_1 \rangle, \dots, \langle l_k, i_k \rangle\},$$

where $\langle l_1, i_1 \rangle, \dots, \langle l_k, i_k \rangle \prec_{s.M_i} \langle l, i \rangle$. Thus, by Lemma 4(a,b),

$$\langle l_\nu, i_\nu, i_\nu \rangle \sqsubseteq \langle l_\nu, i_\nu, i \rangle \sqsubseteq \langle l, i, i \rangle$$

for $\nu = 1, \dots, k$. □

Proof of Theorem 2 (a). Part 1: Termination. Immediately from the definition of the global stack, for every NODPLL transition $s \rightarrow s'$ we have $s.O = s'.O$ if the transition is based on Learn , Conflict , or Explain , and $s.O \prec s'.O$ if the transition is based on any other rule. The ordering \prec here is that of checkpointed sequences, introduced in Section C.

Assume now the theorem is not true and suppose the execution sequence π in (16) is infinite. From the previous paragraphs, we have

$$s_{\text{init}}.O \preceq s_1.O \preceq s_2.O \preceq \dots \tag{17}$$

The invariant (14) (unique occurrence of literals in local stacks) implies that the set of all possible tuples $s.O$ where s is a NODPLL state is finite. This immediately implies that only finitely many inequalities in (17) can be strict.

Since (17) is thus an eventually constant sequence, BackJump (which necessarily changes $s.O$) can occur only finitely many times in π . Since Conflict applies only when $C = \text{no_cflct}$ and replaces it with a set, and since only BackJump can make $C = \text{no_cflct}$ again, we derive that Conflict also can occur only finitely many times in π . Since there are only finitely many clauses that Learn might add to $\Psi_{\mathbb{B}}$, this rule also occurs finitely many times. Therefore, all but finitely many transitions in π are based on Explain .

Now, for some m , we have that all transitions in the subsequence $s_m \rightarrow s_{m+1} \rightarrow \dots$ of π are based on Explain . By Lemma 5, this yields an infinite

⁷ The guard of Explain_i implies that neither $s.C^*$ nor $s'.C^*$ can be no_cflct , so they can be compared by \sqsubseteq_{mul} .

descending chain $s_m.C^* \sqsupset_{\text{mul}} s_{m+1}.C^* \sqsupset_{\text{mul}} \dots$, contradicting well-foundedness of the multiset ordering. This finishes the proof that NODPLL is terminating. \square

A *decision literal* in a state s is any labeled literal $\langle l, \mathbb{B} \rangle$ that occurs immediately after \square in $s.M_{\mathbb{B}}$. Let M_{dec} be the subsequence of $M_{\mathbb{B}}$ consisting of all decision literals and let $M_{\text{dec}}^{[n]}$ be the sequence of the first n decision literals. We will also need the notation for conflict clauses: if $s.C = \{l_1, \dots, l_k\}$, define $s.C^{\text{cls}} = \bar{l}_1 \vee \dots \vee \bar{l}_k$, and if $s.C = \text{no_cflct}$, define $s.C^{\text{cls}} = \text{true}$. Note that $s.C^{\text{cls}} = \text{false}$ if $s.C = \emptyset$.

Lemma 6. *The following are invariants of NODPLL:*

- (a) $\Psi \models_{\mathcal{T}} \Psi_{\text{init}}$ and $\Psi_{\text{init}} \models_{\mathcal{T}} \Psi$;
- (b) $\Psi_{\text{init}} \models_{\mathcal{T}} C^{\text{cls}}$;
- (c) For every i , $\Psi_{\text{init}}, M_{\text{dec}} \models_{\mathcal{T}} M_i$.

Proof. The initial state obviously satisfies all three conditions.

We prove (a) and (b) by simultaneous induction. If then $s \rightarrow s'$ is a transition based on a rule R of NODPLL, and s satisfies (a) and (b), we prove that s' satisfies these conditions too.

If R is not **Learn**, then $s'.\Psi = s.\Psi$ and so s' satisfies (a) by induction hypothesis. If R is **Learn**, then $s'.\Psi = s.\Psi \cup \{s.C^{\text{cls}}\}$, so the proof that s' satisfies (a) is a simple application of the induction hypotheses (a) and (b).

Now we prove that s' satisfies (b): $\Psi_{\text{init}} \models_{\mathcal{T}} s'.C^{\text{cls}}$. The only non-trivial cases are when R is **Conflict_i** or **Explain_i**, since in all other cases we have either $s'.C = s.C$ or $s'.C = \text{true}$ (the latter happens if R is **BackJump**). When R is **Conflict_i**, the truth of the guard $s.\Psi_i, l_i, \dots, l_k \models_i \text{false}$ directly implies $s.\Psi_i \models_i s'.C^{\text{cls}}$. By induction hypothesis (a) $\Psi_{\text{init}} \models_{\mathcal{T}} s.\Psi_i$. Combining the two facts proves our goal.

Suppose finally R is **Explain_i**. The clause $s'.C^{\text{cls}}$ is obtained from $s.C^{\text{cls}}$ by removing the literal \bar{l} and adding literals $\bar{l}_1, \dots, \bar{l}_k$. We have $s.\Psi_i, l_i, \dots, l_k \models_i l$ from the guard of our rule. We also have $\Psi_{\text{init}} \models_{\mathcal{T}} s.C^{\text{cls}}$ and $\Psi_{\text{init}} \models_{\mathcal{T}} s.\Psi_i$ from the induction hypotheses (b) and (a) respectively. Combining these facts proves that s' satisfies (b) too.

Having proved that (a) and (b) hold for all reachable states, it only remains to prove (c). We will prove that the following generalization of (c) holds for all reachable states.

- (c') For every i and n , $\Psi_{\text{init}}, M_{\text{dec}}^{[n]} \models_{\mathcal{T}} M_i^{[n]}$.

Again, we reason by induction. If R , the rule used in a transition $s \rightarrow s'$, is one of the first six (from **Decide** to **EqPropag**) in Figure 4, it is easy to see by direct examination that s' satisfies (c') if s satisfies it. For the next three rules we have $s'M_i = s.M_i$ (for all i), so there is nothing to prove. Thus, we may assume that the transition $s \rightarrow s'$ is based on **BackJump**. The only fact needed to prove that s' satisfies (c') that is not immediately implied by the induction hypothesis is $\Psi_{\text{init}}, M_{\text{dec}}^{[m]} \models_{\mathcal{T}} \bar{l}$, where m and l are as in the rule **BackJump** in Figure 4. This follows from two facts: (1) $\Psi_{\text{init}} \models_{\mathcal{T}} s.C^{\text{cls}}$, and (2) $\Psi_{\text{init}}, M_{\text{dec}}^{[m]} \models_{\mathcal{T}} l'$ for every

$l' \in s.C - \{l\}$. Fact (1) is part (b) of our lemma, and fact (2) follows from our induction hypothesis, because the guard of **BackJump** implies that $l' \in s.M_{\mathbb{B}}^{[m]}$ for every $l' \in s.C - \{l\}$. \square

Proof of Theorem 2 (a). Part 2: Final States. Suppose s is a final state with $s.C \neq \text{no_cflct}$ and let $s_{\text{init}} \rightarrow s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s$ be a run leading to s . Arguing by contradiction, assume $s.C \neq \emptyset$. Note that $\bigvee_{l \in C} \bar{l} \in \Psi_{\mathbb{B}}$, because otherwise **Learn** would apply to s . We claim that in this situation all elements of C are decision literals. Since no two decision literals can have the same level, this claim implies that **BackJump** applies to s , which is a contradiction.

It remains now to prove our claim: every element $\langle l, i \rangle$ of $s.C$ is a decision literal. By Lemma 5 (a), $\langle l, i \rangle$ occurs in $s.M_j$ for some j , and then by Lemma 4 (b), $\langle l, i \rangle \in s.M_i$. Let n be the largest integer such that $\langle l, i, i \rangle \notin s_n.O$. Let R be the rule used in the transition $s_n \rightarrow s_{n+1}$. By definition of the global stack, we have $s_{n+1}.O = s_n.O^{[m]} + \langle l, i, i \rangle$ (when R is **BackJump**), $s_{n+1}.O = s_n.O + \langle l, i, i \rangle$ (when R is **Decide**), or $s_{n+1}.O = s_n.O + \langle l, i, i \rangle$, in the remaining cases. The “remaining cases” actually consist only of **Infer_i**; the propagation rules change the global stack by adding to it triples $\langle l', i', j' \rangle$ with $i' \neq j'$, and the same is true of the dispatch rules by Corollary 1. If R is **Decide**, we are done. Thus, it only remains to eliminate the possibilities when R is **Infer_i** or **BackJump**.

Note that in all cases $s_{n+1}.O$ must be a prefix of $s.O$, because otherwise there would have been a **BackJump** to the level smaller than the level of $\langle l, i, i \rangle$ in the execution $s_{n+1} \rightarrow \dots \rightarrow s$, and it would have removed $\langle l, i, i \rangle$ from the global stack, contradicting the assumption that $\langle l, i, i \rangle$ occurs in $s_i.O$ for all $i \geq n+1$. Thus, $s_{n+1}.M_i$ is a prefix of $s.M_i$.

Suppose R is **Infer_i**. The guard of **Infer_i** implies $\Psi_i, l_1, \dots, l_k \models_i l$ for some $l_1, \dots, l_k \in s_n.M_i$. Since $s_n.M_i$ is a prefix of $s_{n+1}.M_i$, it is also a prefix of $s.M_i$. This implies that **Explain_i** is enabled at s —a contradiction because s is final.

Suppose now R is **BackJump**: $s_{n+1}.O = s_n.O^{[m]} + \langle l, i, i \rangle$. We have $i = \mathbb{B}$ and $\langle l, i \rangle = \langle \bar{l}_0, \mathbb{B} \rangle$, where $s_n.C = \{\langle l_0, i_0 \rangle, \langle l_1, i_1 \rangle, \dots, \langle l_k, i_k \rangle\}$ and $\text{level } l_0 > k \geq \text{level } l_\nu$ ($\nu = 1, \dots, k$). The levels here are computed with respect to s_n . Thus, $l_1, \dots, l_k \in s_n.M_{\mathbb{B}}$. Moreover, since the backjumping level in the transition $s_n \rightarrow s_{n+1}$ is m , from the guard of **BackJump** we have that the literals l_1, \dots, l_k occur in $(s_n.M_{\mathbb{B}})^{[m]}$ (the level of each is at most m). Since $s_{n+1}.M_{\mathbb{B}} = s_n.M_{\mathbb{B}}^{[m]} + \langle \bar{l}_0, \mathbb{B}, \mathbb{B} \rangle$ and $s_{n+1}.M_{\mathbb{B}}$ is a prefix of $s.M_{\mathbb{B}}$, these literals occur in $s.M_{\mathbb{B}}$ as well, and their occurrences in $s.M_{\mathbb{B}}$ precede the occurrence of \bar{l}_0 . The guard of **BackJump** implies also that the clause $\bar{l}_0 \vee \bar{l}_1 \vee \dots \vee \bar{l}_k$ is in $s_n.\Psi_{\mathbb{B}}$, and so it must be in $s.\Psi_{\mathbb{B}}$ too (any transition that changes $\Psi_{\mathbb{B}}$ makes it larger). The facts just collected immediately imply that **Explain_B** applies to s , which (again) is not possible because s is final. \square

Proof of Theorem 2 (b) (Soundness). This is a special case of Lemma 6(b), since $C^{\text{cls}} = \text{false}$ when $C = \emptyset$. \square

F Completeness of NODPLL

Like the original Nelson-Oppen cooperation algorithm, the system NODPLL is not complete. The original completeness theorem requires the participating theories to be convex. The case (i) of Theorem 5 is the analogous result for NODPLL. Completeness can also be achieved by proxying all equalities between shared variables⁸ with fresh propositional variables [4], leaving it to SAT solver to find the arrangement of shared variables by boolean search. This result is covered by case (ii) of Theorem 5.

Theorem 5 (Completeness). *Suppose all theories \mathcal{T}_i are flexible and a final state with $C = \text{no_cflct}$ is reachable from the initial state generated by the input query $\Phi = \Phi_{\mathbb{B}} \cup \Phi_{\mathbb{E}} \cup \Phi_1 \cup \dots \cup \Phi_n$. Suppose also that one of the following conditions holds:*

- (i) \mathcal{T}_i is convex and $\Phi_i^{\text{card}} = \emptyset$, for all $i \in \{1, \dots, n\}$;
- (ii) for every pair x, y of shared variables of the same type, $\Phi_{\mathbb{E}}$ contains a definitional form $p \Leftrightarrow (x = y)$.

Then Φ is \mathcal{T} -satisfiable.

Proof. By Theorem 1, to prove that Φ is \mathcal{T} -satisfiable, it suffices to find

- an assignment M to variables occurring in $\Phi_{\mathbb{B}}$ such that $M \models \Phi_{\mathbb{B}}$
- an arrangement Δ of other shared variables in Φ such that $M, \Delta \models \Phi_{\mathbb{E}}$

and to prove that

$$(\Phi_i \cup \Delta)^{\text{pure}} \cup \Phi_i^{\text{card}} \cup M \text{ is } \mathcal{T}_i\text{-satisfiable for every } i = 1, \dots, n. \quad (18)$$

Now, suppose we are in a final state s of NODPLL. We will first show how s determines an assignment M and an arrangement Δ . Then we will show that (18) holds if we assume either of the conditions (i), (ii).

In s , as in any final state, $M_{\mathbb{B}}$ contains every variable in $V_{\mathbb{B}}$ or its negation—otherwise the rule **Decide** would apply. Thus, $M_{\mathbb{B}}$ can be seen as a full assignment to propositional variables in $V_{\mathbb{B}}$. Since all variables of $\Psi_{\mathbb{B}}$ are in $V_{\mathbb{B}}$, we must have either $M_{\mathbb{B}} \models \Psi_{\mathbb{B}}$ or $M_{\mathbb{B}} \models \neg\Psi_{\mathbb{B}}$. In the latter case, $M_{\mathbb{B}}$ would have to falsify some clause of $\Psi_{\mathbb{B}}$, so the rule **Conflict_B** would apply, which is not true, since we are in a final state. Thus, $M_{\mathbb{B}} \models \Psi_{\mathbb{B}}$. (To be precise, instead of writing $M_{\mathbb{B}}$ and $\Psi_{\mathbb{B}}$ here, we should have written $s.M_{\mathbb{B}}$ and $s.\Psi_{\mathbb{B}}$.) Since $\Psi_{\mathbb{B}}$ is initially $\Phi_{\mathbb{B}}$ and can only grow with applications of NODPLL rules, the desired relation $M_{\mathbb{B}} \models \Phi_{\mathbb{B}}$ holds.

Now, let δ be the set of all pairs $\langle x, y \rangle$ such that $s.\Psi_{\mathbb{E}}, s.M_{\mathbb{E}} \models x = y$. Clearly, δ is a type-respecting equivalence relation. Let Δ be the arrangement defined by δ . That is, for each pair of shared variables x, y of the same type, Δ contains either $x = y$ or $x \neq y$ depending on whether $\langle x, y \rangle \in \delta$ or not. Recall that $\Phi_{\mathbb{E}}$ consists of formulas of the form $p \Leftrightarrow (x = y)$, where $p \in V_{\mathbb{B}}$ and x, y are shared

⁸ A variable is *shared* if it occurs in at least two of the queries Φ_i . Recall that NODPLL requires that V_i contains all shared variables of Φ_i .

variables. We need to show that $M_{\mathbb{B}}, \Delta \models p \Leftrightarrow (x = y)$ holds. Since $M_{\mathbb{B}}$ is an assignment to $V_{\mathbb{B}}$, we have either $p \in M_{\mathbb{B}}$, or $\bar{p} \in M_{\mathbb{B}}$. In the first case, we must have $p \in M_{\mathbb{E}}$, and in the second case, we must have $\bar{p} \in M_{\mathbb{E}}$ —otherwise, $\text{LitDispatch}_{\mathbb{E}}$ would apply to s . (Note that $p, \bar{p} \in L_{\mathbb{E}}$.) By definition of δ , it follows that $\langle x, y \rangle \in \delta$ in the first case, and $\langle x, y \rangle \notin \delta$ in the second case. By definition of Δ , $(x = y) \in \Delta$ in the first case, and $(x \neq y) \in \Delta$ in the second case. It is now immediate that the desired relation $M_{\mathbb{B}}, \Delta \models p \Leftrightarrow (x = y)$ holds in both cases.

With Δ just defined, we can now rewrite our goal (18) as follows.

$$\Psi_i \cup \Delta \cup M_{\mathbb{B}} \text{ is } \mathcal{T}_i\text{-satisfiable for every } i = 1, \dots, n, \quad (19)$$

where again we have taken the liberty to identify the variables in Φ_i with their generalized namesakes in Ψ_i ; see Section D.4. Note that $(\Phi_i \cup \Delta)^{\text{pure}} = \Phi_i^{\text{pure}} \cup \Delta'$ for the appropriate Δ' . If we wanted to be quite precise, we should write this Δ' instead of Δ in (19).

Assume (19) is not true for some i . Observe first that then $\Psi_i \cup \Delta_i \cup M_{\mathbb{B}}$ is not \mathcal{T}_i -satisfiable either, where Δ_i is the subset of Δ containing the (dis)equalities between variables in V_i . Indeed, if one has a \mathcal{T}_i -model for $\Psi_i \cup \Delta_i \cup M_{\mathbb{B}}$, then this model extends to a model of $\Psi_i \cup \Delta \cup M_{\mathbb{B}}$ by interpreting the non- V_i variables of Δ as arbitrary elements constrained only by equalities and disequalities of $\Delta - \Delta_i$. (It is easy to argue that such extensions exist.)

Notice that for unsatisfiability of $\Psi_i \cup \Delta_i \cup M_{\mathbb{B}}$, propositional literals that occur in $M_{\mathbb{B}}$ but not in Ψ_i are irrelevant. Thus, $\Psi_i \cup \Delta_i \cup M_{\mathbb{B}}^i$ is \mathcal{T}_i -unsatisfiable, where $M_{\mathbb{B}}^i$ is the subsequence of $M_{\mathbb{B}}$ containing only variables in $V_{\mathbb{B}} \cap V_i$ and their negations. Now, every element of $M_{\mathbb{B}}^i$ must also be in M_i , because otherwise the rule LitDispatch would apply. As a consequence, $\Psi_i \cup \Delta_i \cup M_i$ is not \mathcal{T}_i -satisfiable.

At this point, we have to use our assumptions (i) or (ii).

Case 1: Assume (i) holds. Write $\Delta_i = \Delta^+ \cup \Delta^-$, where Δ^+ and Δ^- contain equalities and disequalities respectively. Then $\Psi_i, M_i, \Delta^+ \models_i \neg \Delta^-$. For each definitional form $p \Leftrightarrow \phi$ in Ψ_i , either p or \bar{p} occurs in M_i (because $p \in V_{\mathbb{B}} \cap V_i$ and $M_{\mathbb{B}}$ is an assignment to all variables in $V_{\mathbb{B}}$). Obtain Ψ_i' from Ψ_i by replacing $p \Leftrightarrow \phi$ with ϕ or $\neg \phi$, depending on whether M_i contains p or $\neg p$. Clearly, $\Psi_i, M_i \models_i \Psi_i'$ and $\Psi_i', \Delta^+ \models_i \neg \Delta^-$. Now, $\Psi_i' \cup \Delta^+$ is a set of \mathcal{T}_i -literals (here we use the assumption that Ψ_i contains no cardinality constraints) and Δ^- is a disjunction of equalities. Since \mathcal{T}_i is convex, we must have $\Psi_i' \cup \Delta^+ \models_i x = y$ for some $x, y \in V_i$ such that $(x \neq y) \in \Delta^-$. In other words, $\Psi_i, M_i \models_i x = y$ for some x, y such that $\langle x, y \rangle \notin \delta$. Since Infer_i does not apply, we must have $(x = y) \in M_i$. Then, since EqPropag_i does not apply, we must have $(x = y) \in M_{\mathbb{E}}$. By definition of δ , this implies $\langle x, y \rangle \in \delta$, contradicting the already proved fact $\langle x, y \rangle \notin \delta$. This finishes the proof.

Case 2: Assume (ii) holds. Since $\Psi_i \cup \Delta_i \cup M_i$ is \mathcal{T}_i -unsatisfiable, we have $\Psi_i, M_i \models_i \neg \Delta_i$, and so $\Psi_i, M_i \models_i \neg \phi$ for some $\phi \in \Delta_i$. Now, ϕ is either $x = y$ or $x \neq y$, for some $x, y \in V_i$. By assumption (ii), there is a variable $p \in V_{\mathbb{B}} \cap V_i$ such that $p \Leftrightarrow (x = y)$ is in $\Psi_{\mathbb{E}}$. As we already argued, we have either $p \in M_i$ and $\bar{p} \in M_{\mathbb{E}}$, or $\bar{p} \in M_i$ and $p \in M_{\mathbb{E}}$. Assume first $p \in M_i$; then $(x = y) \in \Delta$

(by definition of Δ), so ϕ must be $(x = y)$, so $\Psi_i, M_i \models_i \neg p$, which implies $\Psi_i, M_i \models_i \text{false}$ and therefore applicability of Conflict_i to our final state. This is a contradiction. In the same way the contradiction is obtained in the remaining case $\bar{p} \in M_i$. \square

Note that in case (i), the proof relies on the full strength of rules Infer_i for inferring equalities between variables. In case (ii), however, the proof relies only on the full strength of the rules Conflict_i and it would go through even if the rules Infer_i ($i = 1, \dots, n$) were removed from the system.

Note also that completeness can be proved on the *per theory* basis, where different theories satisfy different sufficient conditions for completeness. For example, instead of requiring in Theorem 5 that one of the conditions (i), (ii) holds, it suffices to make these requirements per theory. Precisely, we can change the last assumption of Theorem 5 to read as follows: for every $i \in \{1, \dots, n\}$, one of the following conditions holds:

- (i') \mathcal{T}_i is convex and $\Phi_i^{\text{card}} = \emptyset$;
- (ii') for every pair $x, y \in V_i$ of shared variables of the same type, $\Phi_{\mathbb{E}}$ contains a definitional form $p \Leftrightarrow (x = y)$.

The proof above would apply without change.

It is important to note that, in the presence of cardinality constraints, convexity does not guarantee completeness. Here is an example.

Example 3. Let x_1, x_2, x_3, x_4 be variables of type α , and let Φ be the \mathcal{T}_{\times} -query consisting of the cardinality constraint $\alpha \doteq 2$ and all constraints $\langle x_i, x_j \rangle \neq \langle x_k, x_l \rangle$, where (i, j, k, l) is a permutation of $(1, 2, 3, 4)$. It is easy to see that Φ is satisfiable and that in every model $\langle \iota, \rho \rangle$ of Φ , three of the variables x_1, x_2, x_3, x_4 are mapped by ρ to the same element of $\iota(\alpha)$, while the fourth is mapped to the other element of $\iota(\alpha)$ (this set has cardinality two).

Now suppose we have five variables x_i of type α , five disjoint theories \mathcal{T}_i and queries Φ_i such that the variables occurring in Φ_i are x_j , where $j \in \{1, \dots, 5\} \setminus \{i\}$. Suppose also that each Φ_i contains the cardinality constraint $\alpha \doteq 2$ and that—as in the example of \mathcal{T}_{\times} and Φ above—for every i , the query Φ_i is \mathcal{T}_i -satisfiable, but that every model requires the four variables in Φ_i to be mapped to the two elements of the domain set in a 3:1 fashion (three variables mapped to the same element, the fourth to a distinct element).

The union of the queries Φ_1, \dots, Φ_5 is unsatisfiable: there is no partition of a set of five elements that when restricted to any four-element subset produces a 3:1 partition.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
2. C. Barrett. *Checking Validity of Quantifier-free Formulas in Combinations of First-Order Theories*. PhD thesis, Stanford University, 2002.

3. C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Splitting on demand in SAT Modulo Theories. In M. Hermann and A. Voronkov, editors, *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'06), Phnom Penh, Cambodia*, volume 4246 of *Lecture Notes in Computer Science*, pages 512–526. Springer, 2006.
4. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani. Efficient theory combination via boolean search. 204(10):1493–1525.
5. R. Bryant, S. Lahiri, and S. Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In E. Brinksma and K. G. Larsen, editors, *Computer Aided Verification: 14th International Conference, CAV*, volume 2404 of *LNCIS*, pages 78–92. Springer, 2002.
6. S. Conchon and S. Krstić. Strategies for combining decision procedures. *Theoretical Computer Science*, 354(2):187–210, 2006.
7. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
8. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. *Journal of the ACM*, 52(3):365–473, 2005.
9. B. Dutertre and L. de Moura. The YICES SMT solver. Technical report, SRI International, 2006.
10. S. Krstić, A. Goel, J. Grundy, and C. Tinelli. Combined satisfiability modulo parametric theories. Technical report, Oct. 2006. (Available at <ftp://ftp.cs.uiowa.edu/pub/tinelli/papers/KrsGGT-RR-06.pdf>).
11. J. C. Mitchell. *Foundations of Programming Languages*. MIT Press, 1996.
12. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.
13. 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*, 2006. (to appear).
14. C. Tinelli and M. Harandi. A new correctness proof of the Nelson-Oppen combination procedure. In F. Baader and K. U. Schulz, editors, *Frontiers of Combining Systems: 1st International Workshop, FroCoS*, volume 3 of *Applied Logic*, pages 103–120. Kluwer, 1996.