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 succintly 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 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* [9] to the current state-of-the-art *Yices* [10], with notable exceptions such as *UCLID* [6], 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" obbligato—that communicate by exchanging equalities between variables ("the Nelson-Oppen style"[15]). 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 5) 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

© Springer-Verlag Berlin Heidelberg 2007

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

Following the precursor [12] 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.[1] Our main results are the termination, soundness, and completeness theorems for our transition system.

Related Work. We were inspired mainly by the work of Nieuwenhuis, Oliveras, and Tinelli [16] on *abstract DPLL* and *abstract DPLL modulo theories*—transition systems that model a DPLL-style SAT solver [8] 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 $DPLL(T_1, \ldots, T_n)$ *architecture*. This architecture is closely related to our system NODPLL (Section 5), 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 $DPLL(T_1, \ldots, T_n)$ 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 [16] and [3] view confict 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 [16] 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 [7], 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], [5]. 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. [5] describes in pseudo-code a particular approach for equality propagation taken by the *MathSAT* solver, which can be modeled in NODPLL; see Section 5.6.

Outline. Section 2 contains (termino)logical background as developed in [12], but divorcing the solver's polymorphic language from *HOL*, to emphasize that

[1] The justification for the presence of non-stably-infinite theories in the Nelson-Oppen framework is studied in recent papers [20,17,4]; in [12], it is shown that the concept of stable-infinity can be dismissed altogether.

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

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}.

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 procedure in the context of parametric theories—and give a suitable form of the non-deterministic Nelson-Oppen combination theorem of [12]. Section 4 is a quick rendition of the core DPLL algorithm as a transition system covering the essential features of modern SAT solvers. Section 5 contains the description of our main transition system for modeling combined SMT solvers, the basic correctness results for it, 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 a brief overview of the syntax and an informal sketch of semantics, referring to [12] 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 $F(\sigma_1, \ldots, \sigma_n)$, where $F \in O$ has arity n and $\sigma_i \in \mathsf{Tp}_O$.

A TYPE INSTANTIATION is a finite map from type variables to types. For any type σ and type instantiation $\theta = [\sigma_1/\alpha_1, \ldots, \sigma_n/\alpha_n]$, $\theta(\sigma)$ denotes the simultaneous substitution of every occurrence of α_i in σ with σ_i. We say that τ is an INSTANCE of σ 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 $(\sigma_0, \ldots, \sigma_n)$. Here, $\sigma_1, \ldots, \sigma_n$ are the argument types of k, and σ_0 is its range type. Constants whose range type is Bool will be called PREDICATES.

We will use the more intuitive notation $k :: [\sigma_1, \dots, \sigma_n] \to \sigma_0$ to indicate the arity of a constant. Moreover, we will write $k\colon [\tau_1, \dots, \tau_n] \to \tau_0$ if there is a type instantiation that maps $\sigma_0, \dots, \sigma_n$ to $\tau_0, \dots, \tau_n$ 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 $\to$ is not a type operator.[2]

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 $k\colon [\tau_1, \dots, \tau_n] \to \tau_0$, 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 define $t' \sqsubseteq t$ to mean that $t' = \theta(t)$ for some θ, and we then say that t' is a TYPE INSTANCE of t and t is a TYPE ABSTRACTION of t'.

For every term t, there exists the MOST GENERAL ABSTRACTION t^{abs} characterized by: (1) $t \sqsubseteq t^{\mathrm{abs}}$; and (2) $t' \sqsubseteq t^{\mathrm{abs}}$ for every t' such that $t \sqsubseteq t'$. The term t^{abs} 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., [13].

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 $\mathsf{List}(E)$ 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 $\mathsf{Array}(I, E)$ of arrays indexed by I with elements in E.

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

The meanings of type operators and constants of a signature together determine a STRUCTURE for that signature. The structure gives meaning to all terms.

[2] In [12], the type and term languages asssociated with a signature were defined as subsets of the higher-order logic, where the function space type operator is primitive and so arities could be seen as types.

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 $\mathsf{Array}(I, E), I, E, I$ respectively) are given, the polymorphic term t becomes a well-defined element of E. In [12], which should be consulted for more details, this element is denoted $[\![t]\!]\langle\iota, \rho\rangle$, where ι and ρ together define an *environment* for t: ι maps the type variables α, β to sets I, E respectively, and ρ maps the variables a, i, x, j to elements of $\mathsf{Array}(I, E), I, E, I$ respectively.

As a boring exercise, the reader may furnish the signatures in Figure 1 with meanings of their type operators and constants, thus obtaining definitions of structures $\mathcal{T}_{\mathsf{Eq}}, \mathcal{T}_{\mathsf{UF}}, \mathcal{T}_{\mathsf{Int}}, \mathcal{T}_{\times}, \mathcal{T}_{\mathsf{Array}}, \mathcal{T}_{\mathsf{List}}$.

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 $[\![\phi]\!]\langle\iota, \rho\rangle = \mathsf{true}$ for some environment $\langle\iota, \rho\rangle$; this environment then is called a MODEL of ϕ. We also say that ϕ is VALID if $\neg\phi$ is unsatisfiable. Validity is denoted $\models_{\mathcal{T}} \phi$, and $\phi_1, \ldots, \phi_n \models_{\mathcal{T}} \phi$ is an abbreviation for $\models_{\mathcal{T}} \phi_1 \wedge \cdots \wedge \phi_n \supset \phi$. 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 $k\, t_1 \ldots t_n$, where k is a predicate. A Σ-LITERAL is an atomic formula or its negation. A CLAUSE is a disjunction of literals. A QUERY is a conjunction of formulas. Clauses containing the same literals in different order are considered equal. (We think of clauses and queries as sets of literals and formulas respectively.) 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 $\alpha \doteq n$, where α is a type variable and n is a positive integer; an enviroment $\langle\iota, \rho\rangle$ satisfies this constraint if α is in the domain of ι and the cardinality of the set $\iota(\alpha)$ is n.

Combining Structures. Two signatures are DISJOINT if the only type operators and constants they share are those of Σ_{Eq}. If $\mathcal{T}_1, \ldots, \mathcal{T}_n$ are structures with pairwise disjoint signatures, then there is a well-defined *sum structure* $\mathcal{T} = \mathcal{T}_1 + \cdots + \mathcal{T}_n$; 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.

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.

In practice, 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 $Q = \Phi_0 \cup \{p_1 \Leftrightarrow \phi_1, \ldots, p_n \Leftrightarrow \phi_n\}$, 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 $Q_M = \{\phi'_1, \ldots, \phi'_n\}$ 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.

Parametricity. There is uniformity in the way "polymorphic" functions like cons compute their results—a consequence of the fact that the definition of cons_E takes the set E as a parameter, making no assumptions about it. Precisely pinning down this uniformity concept is somewhat tricky and we content ourselves with definitions of *parametric type operators* and *parametric constants* that are most convenient for our purposes. They are needed for proper understanding of Theorem 1 below, but not for much else in this paper. Thus, the reader may safely proceed with only a cursory reading of the rest of this section.

Recall first that a relation between two sets A and B is a PARTIAL BIJECTION if it can be seen as a bijection between a subset of A and a subset of B. Define an n-ary set function F to be PARAMETRIC if it is functorial on partial bijections. This means that given any partial bijections $f_i\colon A_i \leftrightarrow B_i$, where $i = 1,\dots,n$, there exists a partial bijection $F(f_1,\dots,f_n)\colon F(A_1,\dots,A_n) \leftrightarrow F(B_1,\dots,B_n)$; moreover, there is a requirement that the identity and composition be preserved. That is, $F(id_{A_1},\dots,id_{A_n}) = id_{F(A_1,\dots,A_n)}$ and $F(g_1 \circ f_1,\dots,g_n \circ f_n) = F(g_1,\dots,g_n) \circ F(f_1,\dots,f_n)$, where $A_i \overset{f_i}{\leftrightarrow} B_i \overset{g_i}{\leftrightarrow} C_i$.

Consider a structure whose type operators are all parametric in the above sense and let $k :: [\sigma_1,\dots,\sigma_n] \to \sigma_0$ be a constant of this structure. Observe that if $\alpha_1,\dots,\alpha_m$ are all type variables that occur in the types σ_i, then any interpretation ι of type variables (that is, an assignment, for each i, of a set A_i to α_i) interprets each type σ_j as a set, say S_j, and interprets k as a function $k_\iota\colon S_1 \times \cdots \times S_n \to S_0$. Suppose now we have two interpretations ι, ι' for type variables, the first just as above, and the second with A'_i and S'_j in place of A_i and S_j. Suppose also that $f_i : A_i \leftrightarrow A'_i$ are partial bijections. Since the type operators of our structure are assumed parametric, there are induced partial bijections $g_j\colon S_j \leftrightarrow S'_j$, for $j = 0,\dots,n$. We say that the constant k is PARAMETRIC if in this situation we have $g_0(k_\iota(x_1,\dots,x_n)) = k_{\iota'}(g_1(x_1),\dots,g_n(x_n))$ for every $x_1 \in \mathtt{dom}(f_1),\dots,x_n \in \mathtt{dom}(f_n)$.

Finally, a PARAMETRIC STRUCTURE is required to have all type operators and all constants parametric. Our example structures $\mathcal{T}_{\mathsf{Eq}}, \mathcal{T}_{\mathsf{UF}}, \mathcal{T}_{\mathsf{Int}}, \mathcal{T}_{\times}, \mathcal{T}_{\mathsf{Array}}, \mathcal{T}_{\mathsf{List}}$, with the notable exception of $\mathcal{T}_{\mathsf{UF}}$, are all parametric; so are the structures describing sets, multisets, and arbitrary algebraic datatypes [12].

We should note that the well-known concept of REYNOLDS PARAMETRICITY in programming languages [18] is neither weaker nor stronger than the concept we are using. In particular, $\mathcal{T}_{\mathsf{UF}}$ is Reynolds parametric. See [12].

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 $\{y^{\mathsf{Int}} = f^{\mathsf{Int}\Rightarrow\mathsf{Int}}\, x^{\mathsf{Int}}, u^{\mathsf{Int}} = f^{\mathsf{Int}\Rightarrow\mathsf{Int}}\, z^{\mathsf{Int}}\}$ 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 $\mathcal{T}_{\mathsf{UF}}$. 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 $\mathcal{T}_{\mathsf{UF}}$-query; however, while being sound, this transformation may compromise completeness. Take the example

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

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

3.1 Solving Semipure Queries

Let us now repeat the above in general terms. Suppose $\Sigma = \Sigma_1 + \cdots + \Sigma_n$ is a sum of pairwise disjoint signatures $\Sigma_i = \langle O_i \mid K_i \rangle$ and $\mathcal{T} = \mathcal{T}_1 + \cdots + \mathcal{T}_n$ is the corresponding sum of theories. Recall that a Σ-term is i-PURE if it is a Σ_i-term. Ignoring "impurity" at the type level, define a Σ-term to be i-SEMIPURE if it can be generated using only constants from K_i. Also, let STRICTLY i-SEMIPURE terms be those that are i-semipure and have no occurrences of logical constants. For example, the queries Φ_1 and Φ_2 above are semipure, while Φ is pure.

Every i-semipure term t has the best PURE ABSTRACTION t^{pure} defined to be the most general abstraction of t with respect to the signature Σ_i. Thus, for every semipure query Φ there exists an essentially unique pure abstraction Φ^{pure}. For example, $\Phi_1^{\mathrm{pure}} = \Phi_2^{\mathrm{pure}} = \Phi$, where the queries are from (1).

Note that Σ-terms without occurrences of non-logical constants are i-semipure for every i. For such a term t, we define t^{pure} to be the $\mathcal{T}_{\mathsf{Eq}}$-pure abstraction of t. For example, $(x^{\mathsf{Int}} = y^{\mathsf{Int}} \wedge u^{\mathsf{Int}\times\mathsf{Int}} \neq v^{\mathsf{Int}\times\mathsf{Int}})^{\mathrm{pure}} = (x^{\alpha} = y^{\alpha} \wedge u^{\beta} \neq v^{\beta})$.

If the query Φ is semipure and θ is a type instantiation such that $\theta(\Phi^{\mathrm{pure}}) = \Phi$, denote by Φ^{card} the set of cardinality constraints $\alpha \doteq n$, where $\alpha \in \mathtt{dom}(\theta)$ 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* [12]. Hence, a strong solver for i-pure queries suffices to solve i-semipure queries.

3.2 Purification

Let $\mathcal{T}$ and $\mathcal{T}_1, \ldots, \mathcal{T}_n$ be as above. Given any input $\mathcal{T}$-query, we can purify it by introducing proxy variables and definitional 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:

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

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 [12] under the assumption that non-logical constants do not take arguments of boolean type.[3]

Partition now Φ into subsets $\Phi_{\mathbb{B}}, \Phi_{\mathbb{E}}, \Phi_1, \ldots, \Phi_n$, where $\Phi_{\mathbb{B}}$ contains the propositional clauses 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 $\Phi_{\mathbb{B}} = \{p, q\}$, $\Phi_{\mathbb{E}} = \{p \Leftrightarrow y = x\}$, $\Phi_{\mathsf{UF}} = \{y = f(x), u = f(z)\}$, $\Phi_{\mathsf{Int}} = \{z = 2x - y, q \Leftrightarrow u > x\}$. For readability we omit type superscripts on variables, but see Figure 4 below where this example can be seen in its full typed glory.

3.3 Nelson-Oppen

Let $\Phi = \Phi_{\mathbb{B}} \cup \Phi_{\mathbb{E}} \cup \Phi_1 \cup \cdots \cup \Phi_n$ be as above and let us call a variable SHARED if it occurs in at least two of the queries $\Phi_{\mathbb{B}}, \Phi_{\mathbb{E}}, \Phi_1, \ldots, \Phi_n$. For a set V of variables, define an ARRANGEMENT on V to be 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 [12]. It is the basis of the non-deterministic Nelson-Oppen procedure in the style of Tinelli-Harandi [19], but for parametric theories.

[3] 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 equisatifiable form Φ also requires that all occurrences of ite be compiled away by replacing $z = \mathsf{ite}(p, x, y)$ with the equivalent $(p \supset z = x) \wedge (\bar{p} \supset z = y)$.

Theorem 1. *Suppose each of the theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$ is either $\mathcal{T}_{\mathsf{UF}}$ or is parametric. Let M be an assignment*[4] *to shared propositional variables and let Δ be an arrangement of all remaining shared variables. Then: $\Phi \cup \Delta \cup M$ is $\mathcal{T}$-satisfiable if and only if (with the convention that $\mathcal{T}_{\mathbb{E}}$ stands for $\mathcal{T}_{\mathsf{Eq}}$)*

– *$\Phi_{\mathbb{B}} \cup M$ is satisfiable;*
– *$(\Phi_i \cup \Delta)^{\mathrm{pure}} \cup \Phi_i^{\mathrm{card}} \cup M$ is $\mathcal{T}_i$-satisfiable, for every $i \in \{\mathbb{E}, 1, \ldots, n\}$.*

4 DPLL with Conflict Analysis

We define a transition system DPLL that models the operation of a DPLL-style SAT solver, including conflict analysis. In Section 5 it will be extended with rules for Nelson-Oppen cooperation of multiple theories.

The only parameter of DPLL 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 $\mathsf{no\_cflct}$.

The "input" to DPLL is an arbitrary set of clauses Ψ_{init}, modeled as an initial state in which $\Psi = \Psi_{\mathrm{init}}$, $M = []$, and $C = \mathsf{no\_cflct}$. 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 of l' in M. Note that M can be written uniquely in the form $M = M^{\langle 0 \rangle} + \square + M^{\langle 1 \rangle} + \square + \cdots + \square + M^{\langle d \rangle}$, where $+$ denotes sequence concatenation and $\square$ does not occur in any $M^{\langle i \rangle}$. The superscripts indicate "decision levels" and $M^{[m]} = M^{\langle 0 \rangle} + \square + \cdots + \square + M^{\langle m \rangle}$ is the prefix of M up to decision level m. A literal can occur at most once in M; we write $\mathsf{level}\ l = i$ if l occurs in $M^{\langle i \rangle}$. Finally, the number k occurring in the rules is an arbitrary non-negative integer.

An example run of DPLL is given in Figure 3. The correctness of the system is expressed by the following theorem, proved in the appendix.

Theorem 2 (Correctness). *All runs of* DPLL *are finite. If, initialized with the set of clauses Ψ_{init},* DPLL *terminates in the state $\langle \Psi, M, C \rangle$, then: (a) $C = \mathsf{no\_cflct}$ or $C = \varnothing$; (b) If $C = \varnothing$ then Ψ_{init} is unsatisfiable; (c) If $C = \mathsf{no\_cflct}$, then M is a model for Ψ_{init}.*

By restricting the set of behaviors of NODPLL, we can obtain more accurate models of modern SAT solvers. Let $\mathsf{Explain\_uip}$ be the rule obtained by strengthening the guard of $\mathsf{Explain}$ with the conditions $\forall l' \in C.\ l' \preceq l$ and $\exists l' \in C.\ \mathsf{level}\ l' = \mathsf{level}\ l$ that force the explanation sequence to find the first "unique implication point" [14] and stop when it is has been found. Consider the strategy

[4] 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.

$$\mathsf{Decide} \qquad \frac{l \in L \qquad l, \bar{l} \notin M}{M := M + \square + l}$$

$$\mathsf{UnitPropag} \qquad \frac{l \vee l_1 \vee \cdots \vee l_k \in \Psi \qquad \bar{l}_1, \ldots, \bar{l}_k \in M \qquad l, \bar{l} \notin M}{M := M + l}$$

$$\mathsf{Conflict} \qquad \frac{C = \mathsf{no\_cflct} \qquad \bar{l}_1 \vee \cdots \vee \bar{l}_k \in \Psi \qquad l_1, \ldots, l_k \in M}{C := \{l_1, \ldots, l_k\}}$$

$$\mathsf{Explain} \qquad \frac{l \in C \qquad l \vee \bar{l}_1 \vee \cdots \vee \bar{l}_k \in \Psi \qquad l_1, \ldots, l_k \prec l}{C := C \cup \{l_1, \ldots, l_k\} \smallsetminus \{l\}}$$

$$\mathsf{Learn} \qquad \frac{C = \{l_1, \ldots, l_k\} \qquad \bar{l}_1 \vee \cdots \vee \bar{l}_k \notin \Psi}{\Psi := \Psi \cup \{\bar{l}_1 \vee \cdots \vee \bar{l}_k\}}$$

$$\mathsf{BackJump} \qquad \frac{C = \{l, l_1, \ldots, l_k\} \qquad \bar{l} \vee \bar{l}_1 \vee \cdots \vee \bar{l}_k \in \Psi \qquad \begin{matrix}\mathsf{level}\ l > m \geq \mathsf{level}\ l_i \\ (i = 1, \ldots, k)\end{matrix}}{C := \mathsf{no\_cflct} \qquad M := M^{[m]} + \bar{l}}$$

Fig. 2. Rules of DPLL

$$(((\mathsf{Conflict}\,;\ \mathsf{Explain\_uip}^*\ ;\ [\mathsf{Learn}\,;\ \mathsf{BackJump}]) \parallel \mathsf{UnitPropag})^*\ ;\ [\mathsf{Decide}])^* \qquad (2)$$

where $\parallel$ denotes the non-deterministic choice, α^* means "apply α as long as possible", and $[\alpha]$ means "apply α once if possible". Note that the rule Conflict is triggered by a clause all of whose literals are asserted false in M, and UnitPropag is triggered by a clause in which all but one literal is asserted false in M. To turn (2) into a deterministic algorithm, one must specify the search for the trigger clause. This is what the "two-watched-literals" scheme is for; it ensures that trigger clauses are quickly found after each Decide step. When Decide is to be applied, the choice of the decision literal is based on some heuristics, the most popular being VSIDS of [14]. As for the "non-chronological backtracking" in BackJump, the backjump level m is normally taken to be the minimum possible, i.e., the largest of the numbers $\mathsf{level}\ l_i$ $(i = 1, \ldots, k)$. With the exception of restarts and clause forgetting (easily modeled, harder to tune; see Figure 6), the above gives a rather complete top-level picture of *Chaff* [14]. One can also prove that for the strategy (2) the guard of Learn is automatically satisfied, justifying the fact that the implementations do not perform this expensive check.

The top-level *MiniSAT* [11] looks almost the same; we only need to replace Explain_uip* in (2) with the sequence Explain_uip* ; Explain_min*, where the conflict clause-minimizing rule Explain_min is obtained by strengthening the guard of Explain with the condition $l_1, \ldots, l_m \in C$.

$$
\begin{array}{cccc}
\langle \Psi, [\,], \mathsf{no\_cflct}\rangle & \xrightarrow{\mathsf{Decide}} & \langle \Psi, \Box 1, \mathsf{no\_cflct}\rangle & \xrightarrow{\mathsf{UnitPropag}} \\
\langle \Psi, \Box 12, \mathsf{no\_cflct}\rangle & \xrightarrow{\mathsf{Decide}} & \langle \Psi, \Box 12\Box 3, \mathsf{no\_cflct}\rangle & \xrightarrow{\mathsf{UnitPropag}} \\
\langle \Psi, \Box 12\Box 34, \mathsf{no\_cflct}\rangle & \xrightarrow{\mathsf{Decide}} & \langle \Psi, \Box 12\Box 34\Box 5, \mathsf{no\_cflct}\rangle & \xrightarrow{\mathsf{UnitPropag}} \\
\langle \Psi, \Box 12\Box 34\Box 56, \mathsf{no\_cflct}\rangle & \xrightarrow{\mathsf{Conflict}} & \langle \Psi, \Box 12\Box 34\Box 56, \{2,5,6\}\rangle & \xrightarrow{\mathsf{Explain}} \\
\langle \Psi, \Box 12\Box 34\Box 56, \{2,5\}\rangle & \xrightarrow{\mathsf{Learn}} & \langle \Psi', \Box 12\Box 34\Box 56, \{2,5\}\rangle & \xrightarrow{\mathsf{BackJump}} \\
\langle \Psi', \Box 12\bar{5}, \mathsf{no\_cflct}\rangle & \xrightarrow{\mathsf{Decide}} & \langle \Psi', \Box 12\bar{5}\Box 3, \mathsf{no\_cflct}\rangle & \xrightarrow{\mathsf{UnitPropag}} \\
\langle \Psi', \Box 12\bar{5}\Box 34, \mathsf{no\_cflct}\rangle & \xrightarrow{\mathsf{Decide}} & \langle \Psi', \Box 12\bar{5}\Box 34\Box \bar{6}, \mathsf{no\_cflct}\rangle &
\end{array}
$$

Fig. 3. A 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 [16]. 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.

5 The Combined Solver

In this section, we define of our Nelson-Oppen-with-DPLL transition system NODPLL, state its correctness results, and discuss some features and extensions. The system can adequately express important design decisions and allows a great deal of freedom for implementation.

The parameters of NODPLL are pairwise disjoint theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$. The "input" to NODPLL is a $(\mathcal{T}_1 + \cdots + \mathcal{T}_n)$-query Φ, purified into $\Phi_{\mathbb{B}} \cup \Phi_{\mathbb{E}} \cup \Phi_1 \cup \cdots \cup \Phi_n$ as in Section 3. Assume that all variables occurring in these formulas have distinct names and let $\mathtt{vars}(\Theta)$ denote the set of names of variables in Θ.

5.1 State, Rules, and Initialization

Let $I = \{\mathbb{B}, \mathbb{E}, 1, \ldots, n\}$. The state variables of NODPLL are the following:

- SHARED VARIABLE SETS V_i for $i \in I$. These are sets of variable names, not necessarily disjoint. For every $i \in I$ define (paraphrasing [5]) the set L_i of INTERFACE LITERALS to consist of variables in $V_i \cap V_{\mathbb{B}}$ and their negations, and also equalities of the form $x = y$, where $x, y \in V_i \smallsetminus V_{\mathbb{B}}$.
- LOCAL CONSTRAINTS Ψ_i for $i \in I$. Here, $\Psi_{\mathbb{B}}$ is a set of propositional clauses and Ψ_i for $i \in \{\mathbb{E}, 1, \ldots, n\}$ is a set of (pure) $\mathcal{T}_i$-formulas and cardinality constraints. We require that $\mathtt{vars}(\Psi_{\mathbb{B}}) \subseteq V_{\mathbb{B}}$ and that $\mathtt{vars}(\Psi_i) \cap \mathtt{vars}(\Psi_j) \subseteq V_i \cap V_j \cap V_{\mathbb{E}}$ for distinct $i, j \neq \mathbb{B}$.
- LOCAL STACKS (checkpointed sequences) M_i for $i \in I$. Any element of M_i is either $\Box$ 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 $\mathsf{no\_cflct}$.

The input query Φ defines the INITIAL STATE s^{Φ}_{init}, in which: $\Psi_{\mathbb{B}} = \Phi_{\mathbb{B}}$; $\Psi_i = \Phi_i^{\mathrm{pure}} \cup \Phi_i^{\mathrm{card}}$ for $i \neq \mathbb{B}$; $M_i = [\,]$ for every i; $C = \mathsf{no\_cflct}$; $V_{\mathbb{B}} = \mathtt{vars}(\Phi_{\mathbb{B}})$; $V_i = \bigcup_{j \neq i} \mathtt{vars}(\Phi_i) \cap \mathtt{vars}(\Phi_j)$ for $i \neq \mathbb{B}, \mathbb{E}$; and $V_{\mathbb{E}} = (\mathtt{vars}(\Phi_{\mathbb{E}}) \cap \mathtt{vars}(\Phi_{\mathbb{B}})) \cup \bigcup_{i \neq \mathbb{B}} (V_i \smallsetminus V_{\mathbb{B}})$. An example is given in Figure 4.

The transitions of NODPLL are defined by the rules in Figure 5. The following two paragraphs explain the additional notation used in these rules.

$$(a) \quad \Phi = \{f^{\mathsf{Int}\Rightarrow\mathsf{Int}}(x^{\mathsf{Int}}) = x^{\mathsf{Int}}, f^{\mathsf{Int}\Rightarrow\mathsf{Int}}(2x^{\mathsf{Int}} - f^{\mathsf{Int}\Rightarrow\mathsf{Int}}(x^{\mathsf{Int}})) > x^{\mathsf{Int}}\}$$

$$(b) \quad \begin{array}{ll} \Phi_{\mathbb{B}} = \{p^{\mathsf{Bool}}, q^{\mathsf{Bool}}\} & \Phi_{\mathsf{Int}} = \{z^{\mathsf{Int}} = 2x^{\mathsf{Int}} - y^{\mathsf{Int}}, q^{\mathsf{Bool}} \Leftrightarrow u^{\mathsf{Int}} > x^{\mathsf{Int}}\} \\ \Phi_{\mathbb{E}} = \{p^{\mathsf{Bool}} \Leftrightarrow (y^{\mathsf{Int}} = x^{\mathsf{Int}})\} & \Phi_{\mathsf{UF}} = \{y^{\mathsf{Int}} = f^{\mathsf{Int}\Rightarrow\mathsf{Int}}(x^{\mathsf{Int}}), u^{\mathsf{Int}} = f^{\mathsf{Int}\Rightarrow\mathsf{Int}}(z^{\mathsf{Int}})\} \end{array}$$

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

Fig. 4. Initialization of NODPLL. *(a)* the input query Φ from Example 1, *(b)* the result of purifying Φ, *(c)* generated local constraints for NODPLL. The shared variable sets are $V_{\mathbb{B}} = \{p, q\}$, $V_{\mathsf{UF}} = \{x, y, z, u\}$, and $V_{\mathsf{Int}} = \{x, y, z, u, q\}$. Note that 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.

We write $C^{\flat}$ 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 $\mathcal{T}_{\mathsf{Eq}}$. Note that all literals occurring in the rules contain untyped (shared) variables and so there is an abuse of notation when we write $\Psi_i, l_1, \ldots, l_k \models_i l$ in the rule $\mathsf{Explain}_i$. Of course, this entailment should be understood as $\Psi_i, l'_1, \ldots, l'_k \models_i l'$ where each primed literal is obtained by replacing the untyped variables occurring in it with the equally named typed variable of Ψ_i. (Recall that the variables of Ψ_i have the same names as variables of Φ_i, but possibly more general types.) The same convention is used in the rules Infer_i and $\mathsf{Conflict}_i$.

It is easy to see that *all local stacks have the same number of occurrences of* $\Box$. Thus, each M_i can be written as $M_i = M_i^{\langle 0\rangle} + \Box + M_i^{\langle 1\rangle} + \Box + \cdots + \Box + M_i^{\langle d\rangle}$, where d is the CURRENT DECISION LEVEL and $\Box$ does not occur in any $M_i^{\langle k\rangle}$. Note that some of the sequences $M_i^{\langle k\rangle}$ may be empty; however, $M_{\mathsf{Bool}}^{\langle k\rangle}$ is nonempty for $1 \leq k \leq d$ and its first element is called the kth DECISION LITERAL. The rule $\mathsf{Explain}$ uses the notation $\langle l, j\rangle \prec_{M_i} \langle l', j'\rangle$; by definition, this means that both labeled literals occur in M_i and that the occurrence of $\langle l, j\rangle$ precedes the occurrence of $\langle l', j'\rangle$. For correctness of this definition, we need to know that *in any local stack, any literal can occur at most once.* This invariant is proved in the appendix. Finally, the function level used in the $\mathsf{BackJump}$ rule is defined only for literals that occur in $M_{\mathbb{B}}$; we have $\mathsf{level}\, l = k$ if l occurs in $M_{\mathbb{B}}^{\langle k\rangle}$.

This completes the definition of the system NODPLL. As with DPLL, to check the satisfiability of a formula Φ, we can start with the state s_{init}^{Φ} and apply the rules of NODPLL in arbitrary order; if we end up with a final state in which $C = \mathsf{no\_cflct}$, then Φ is satisfiable; otherwise, the final state will have $C = \varnothing$ and Φ is unsatisfiable.

Example 2. If NODPLL is initialized with $\Psi_{\mathbb{B}}, \Psi_{\mathbb{E}}, \Psi_{\mathsf{UF}}, \Psi_{\mathsf{Int}}$ from Figure 4, the first 13 steps could be (1) $\mathsf{Infer}_{\mathbb{B}}$, (2) $\mathsf{Infer}_{\mathbb{B}}$, (3) $\mathsf{LitDispatch}_{\mathbb{E}}$, (4) $\mathsf{LitDispatch}_{\mathsf{Int}}$,

Decide
$$\frac{l \in L_{\mathbb{B}} \qquad l, \bar{l} \notin M_{\mathbb{B}}}{M_{\mathbb{B}} := M_{\mathbb{B}} + \Box + \langle l, \mathbb{B}\rangle \qquad M_i := M_i + \Box \quad (\text{all } i \neq \mathbb{B})}$$

Infer$_i$
$$\frac{l \in L_i \qquad l, \bar{l} \notin M_i \qquad \Psi_i, M_i \models_i l}{M_i := M_i + \langle l, i\rangle}$$

LitDispatch$_i$
$(i \neq \mathbb{B})$
$$\frac{l \in L_i \qquad \langle l, j\rangle \in M_{\mathbb{B}} \qquad l \notin M_i}{M_i := M_i + \langle l, j\rangle}$$

EqDispatch$_i$
$(i \neq \mathbb{E}, \mathbb{B})$
$$\frac{x, y \in V_i \qquad \langle x = y, j\rangle \in M_{\mathbb{E}} \qquad x = y \notin M_i}{M_i := M_i + \langle x = y, j\rangle}$$

LitPropag$_i$
$(i \neq \mathbb{B})$
$$\frac{l \in L_{\mathbb{B}} \qquad \langle l, i\rangle \in M_i \qquad l, \bar{l} \notin M_{\mathbb{B}}}{M_{\mathbb{B}} := M_{\mathbb{B}} + \langle l, i\rangle}$$

EqPropag$_i$
$(i \neq \mathbb{E}, \mathbb{B})$
$$\frac{x, y \in V_{\mathbb{E}} \qquad \langle x = y, i\rangle \in M_i \qquad x = y \notin M_{\mathbb{E}}}{M_{\mathbb{E}} := M_{\mathbb{E}} + \langle x = y, i\rangle}$$

Conflict$_i$
$$\frac{C = \text{no\_cflct} \qquad \langle l_1, i_1\rangle, \ldots, \langle l_k, i_k\rangle \in M_i \qquad \Psi_i, l_1, \ldots, l_k \models_i \text{false}}{C := \{\langle l_1, i_1\rangle, \ldots, \langle l_k, i_k\rangle\}}$$

Explain$_i$
$$\frac{\langle l, i\rangle \in C \qquad \langle l_1, i_1\rangle, \ldots, \langle l_k, i_k\rangle \prec_{M_i} \langle l, i\rangle \qquad \Psi_i, l_1, \ldots, l_k \models_i l}{C := C \cup \{\langle l_1, i_1\rangle, \ldots, \langle l_k, i_k\rangle\} \setminus \{\langle l, i\rangle\}}$$

Learn
$$\frac{C^\flat = \{l_1, \ldots, l_k\} \qquad C^\flat \subseteq L_{\mathbb{B}} \qquad \bar{l}_1 \vee \cdots \vee \bar{l}_k \notin \Psi_{\mathbb{B}}}{\Psi_{\mathbb{B}} := \Psi_{\mathbb{B}} \cup \{\bar{l}_1 \vee \cdots \vee \bar{l}_k\}}$$

BackJump
$$\frac{C^\flat = \{l, l_1, \ldots, l_k\} \qquad \bar{l} \vee \bar{l}_1 \vee \cdots \vee \bar{l}_k \in \Psi_{\mathbb{B}} \qquad \begin{matrix}\text{level } l > m \geq \text{level } l_i \\ (i = 1, \ldots, k)\end{matrix}}{C := \text{no\_cflct} \qquad M_{\mathbb{B}} := M_{\mathbb{B}}^{[m]} + \langle \bar{l}, \mathbb{B}\rangle \qquad M_i := M_i^{[m]} \quad (\text{all } i \neq \mathbb{B})}$$

Fig. 5. Rules of NODPLL

(5) Infer$_{\mathbb{E}}$, (6) EqDispatch$_{\text{Int}}$, (7) Infer$_{\text{Int}}$, (8) EqPropag$_{\text{Int}}$, (9) EqDispatch$_{\text{UF}}$, (10) EqDispatch$_{\text{UF}}$, (11) Infer$_{\text{UF}}$, (12) EqPropag$_{\text{UF}}$, (13) EqDispatch$_{\text{Int}}$, with these rules modifying the local stacks as follows:

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

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

$$\begin{array}{ll}
\text{Forget} & \dfrac{C = \mathsf{no\_cflct} \qquad \phi \in \Psi_{\mathbb{B}} \qquad \Psi_{\mathbb{B}} \setminus \{\phi\} \models \phi}{\Psi_{\mathbb{B}} := \Psi_{\mathbb{B}} - \{\phi\}} \\[2ex]
\text{Restart} & \dfrac{C = \mathsf{no\_cflct}}{M_i := M_i^{[0]} \quad (\text{all } i)} \\[2ex]
\text{ThLearn}_i & \dfrac{l_1, \dots, l_k \in L_{\mathbb{B}} \qquad \Psi_i \models_i l_1 \vee \cdots \vee l_k}{\Psi_{\mathbb{B}} = \Psi_{\mathbb{B}} \cup \{l_1 \vee \cdots \vee l_k\}}
\end{array}$$

Fig. 6. Additional rules for NODPLL

Conflict$_{\mathsf{Int}}$, Explain$_{\mathsf{UF}}$, Explain$_{\mathsf{Int}}$, Explain$_{\mathbb{E}}$, Explain$_{\mathbb{B}}$,Explain$_{\mathbb{B}}$ that transform C as follows: no_cflct $\to \{q, y = u\} \to \{q, z = x\} \to \{q, y = x\} \to \{q, p\} \to \{q\} \to \varnothing$.

An imlementation of NODPLL would require theory solvers $\mathbf{S}_{\mathbb{B}}, \mathbf{S}_{\mathbb{E}}, \mathbf{S}_1, \dots, \mathbf{S}_n$. Each solver $\mathbf{S}_i$ would be responsible for maintaining its local stack M_i, local constraint set Ψ_i, and for the implementation of the rules Infer$_i$, LitPropag$_i$, EqPropag$_i$, Conflict$_i$, Explain$_i$, BackJump. The SAT solver $\mathbf{S}_{\mathbb{B}}$ would additionally implement the rules Decide, LitDispatch$_i$, Learn; and the pure equality solver $\mathbf{S}_{\mathbb{E}}$ would additionally implement the rules EqDispatch$_i$. In addition, a central controller $\mathbf{C}$ would be needed to manage the global conflict set C. With a little effort to ensure that explanations from theory solvers are expressed in terms of SAT literals, the conflict set could alternatively be handled by the SAT solver.

5.2 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. This intricate dynamics makes NODPLL significantly more complex than DPLL, but the basic correctness properties still hold.

Theorem 3 (Termination). *Every run of* NODPLL *is finite and ends in a state where* $C =$ no_cflct *or* $C = \varnothing$*.*

Theorem 4 (Soundness). *If a final state with* $C = \varnothing$ *is reachable, then the input query is* $\mathcal{T}$*-unsatisfiable.*

Completeness results for NODPLL are derived from Theorem 1. Note that Theorem 1 immediately implies a complete decision procedure: enumerate all assignments M to shared propositional variables and all arrangements Δ for other shared variables and see if there is a pair M, Δ that is consistent with all local constraints Ψ_i. Instead of this inefficient blind search, NODPLL progressively builds M as $M_{\mathbb{B}}$ and Δ as the set of all equalities implied by $\Psi_{\mathbb{E}} \cup M_{\mathbb{E}}$. Thus, when an execution of NODPLL started at s_{init}^{Φ} reaches a final state where $C = \mathsf{no\_cflct}$, if we can prove that the accumulated M and Δ are consistent with the local constraints, it would follow that Φ is satisfiable. However, if a participating theory $\mathcal{T}_i$ is not convex, it may happen that Ψ_i implies $x = y \vee u = v$, while neither $x = y$ nor $u = v$ is implied by any local constraints; thus, Δ contains $x \neq y$ and $u \neq v$ and so is not consistent with Ψ_i. It is well known that convexity guarantees completeness for the original basic Nelson-Oppen cooperation algorithm [15], and case (i) of Theorem 5 is the analogous result for NODPLL. Completeness can also be achieved by proxying all equalities between shared variables with fresh propositional variables ([5], see Section 5.6 below), leaving it to the SAT solver to find the arrangement of shared variables by purely boolean search. This result is covered by case (ii) of Theorem 5.

Theorem 5 (Completeness). *Suppose each of the theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$ is either $\mathcal{T}_{\mathsf{UF}}$ or is parametric. Suppose a final state with $C = \mathsf{no\_cflct}$ is reachable from the initial state generated by the input query $\Phi = \Phi_{\mathbb{B}} \cup \Phi_{\mathbb{E}} \cup \Phi_1 \cup \cdots \cup \Phi_n$. Suppose also that one of the following conditions holds:*

(i) *$\mathcal{T}_i$ is convex and $\Phi_i^{\text{card}} = \varnothing$, for all $i \in \{1, \ldots, n\}$;*
(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.

Theorems 3, 4 and 5 are proved in the appendix.

5.3 Strengthening Propositional Rules

Clearly, DPLL is a subsystem of NODPLL, but its rules UnitPropag, Conflict and Explain are less permissive than the corresponding rules $\mathsf{Infer}_{\mathbb{B}}$, $\mathsf{Conflict}_{\mathbb{B}}$ and $\mathsf{Explain}_{\mathbb{B}}$ of NODPLL. If we change these three rules of NODPLL with the equally named rules in Figure 7, then the correspondence would be exact. This change would not affect NODPLL in any significant way. In particular, the proof of correctness properties of NODPLL given in the appendix would apply verbatim to the modified system.

5.4 Why Are the Literals Labeled?

Every literal that gets added to any of the local stacks is a logical consequence of the decision literals and the local constraints Ψ_i. 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 literal explanation process will terminate (with a conflicting set of decision literals).

$$\mathsf{Infer}_{\mathbb{B}} \quad \frac{l \in L_{\mathbb{B}} \qquad l, \bar{l} \notin M_{\mathbb{B}} \qquad l \vee l_1 \vee \cdots \vee l_k \in \Psi \qquad \bar{l}_1, \ldots, \bar{l}_k \in M_{\mathbb{B}}}{M_{\mathbb{B}} := M_{\mathbb{B}} + \langle l, \mathbb{B} \rangle}$$

$$\mathsf{Conflict}_{\mathbb{B}} \quad \frac{C = \mathsf{no\_cflct} \qquad \langle l_1, i_1 \rangle, \ldots, \langle l_k, i_k \rangle \in M_{\mathbb{B}} \qquad \bar{l}_1 \vee \cdots \vee \bar{l}_k \in \Psi_{\mathbb{B}}}{C := \{\langle l_1, i_1 \rangle, \ldots, \langle l_k, i_k \rangle\}}$$

$$\mathsf{Explain}_{\mathbb{B}} \quad \frac{\langle l, \mathbb{B} \rangle \in C \qquad \langle l_1, i_1 \rangle, \ldots, \langle l_k, i_k \rangle \prec_{M_{\mathbb{B}}} \langle l, \mathbb{B} \rangle \qquad l \vee \bar{l}_1 \vee \cdots \vee \bar{l}_k \in \Psi_{\mathbb{B}}}{C := C \cup \{\langle l_1, i_1 \rangle, \ldots, \langle l_k, i_k \rangle\} \smallsetminus \{\langle l, \mathbb{B} \rangle\}}$$

Fig. 7. Rules of DPLL within NODPLL

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 would be too ambiguous and potentially circular—the phenomenon termed "too new explanations" in [16]. As an illustration, consider the five-step execution of NODPLL in Figure 8, 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. We assume that $\bar{l} \vee l' \in \Psi_{\mathbb{B}}$ and $\Psi_2, l' \models_2 l$—the facts responsible for the $\mathsf{Infer}_{\mathbb{B}}$ and Infer_2 steps.

$$\begin{array}{ll} M_{\mathbb{B}}: \; [\,] \xrightarrow{(2)} [l] \xrightarrow{(3)} [l, l'] & \qquad M_{\mathbb{B}}: \; [\,] \xrightarrow{(2)} [\langle l, 1 \rangle] \xrightarrow{(3)} [\langle l, 1 \rangle, \langle l', \mathbb{B} \rangle] \\ M_1: \; [\,] \xrightarrow{(1)} [l] & \qquad M_1: \; [\,] \xrightarrow{(1)} [\langle l, 1 \rangle] \\ M_2: \; [\,] \xrightarrow{(4)} [l'] \xrightarrow{(5)} [l', l] & \qquad M_2: \; [\,] \xrightarrow{(4)} [\langle l', \mathbb{B} \rangle] \xrightarrow{(5)} [\langle l', \mathbb{B} \rangle, \langle l, 2 \rangle] \end{array}$$

Fig. 8. A five-step run: (1) Infer_1, (2) $\mathsf{LitPropag}_1$, (3) $\mathsf{Infer}_{\mathbb{B}}$, (4) $\mathsf{LitDispatch}_2$, (5) Infer_2

Suppose that, as shown on the left in Figure 8, we use non-labeled literals in the local stacks. Assume that sometime in the future a conflict arises, with the conflict set that contains l but not l'. Say, $C = \{l\} \cup D$. Then $\mathsf{Explain}_2$ applies and changes C to $\{l'\} \cup D$. But then $\mathsf{Explain}_{\mathbb{B}}$ applies and changes C back to $\{l\} \cup D$. So the system can repeat explaining l and l' with each other forever.

The same situation, but with labeled literals, is shown on the right in Figure 8: The conflict set C is now $\{\langle l, 2 \rangle\} \cup D$. It becomes $\{\langle l', \mathbb{B} \rangle\} \cup D$ after $\mathsf{Explain}_2$, and then becomes $\{\langle l, 1 \rangle\} \cup D$ after $\mathsf{Explain}_{\mathbb{B}}$. 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 $\mathsf{Explain}_1$ will have to explain $\langle l, 1 \rangle$ with the (in this case, empty) set of literals that were used in the Infer_1 step.

5.5 Basic Extensions

The system NODPLL can be extended with new rules to capture secondary features, often without significantly complicating correctness proofs.

The rules Forget Restart, and ThLearn in Figure 6 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 [16].

Extending NODPLL with the rule ThLearn makes it possible to propagate disjuncions of shared equalities; one can prove, by an argument that goes back to [15], that with this rule the system becomes complete even for non-convex theories.

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 equisatifiable Ψ_i' that may even use fresh variables [7,16] 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].

5.6 Equality Proxying and Propagation

If in Example 2 we had a proxy for $y = u$, say $r \Leftrightarrow y = u$, with this definional form put in both Ψ_{Int} and Ψ_{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.

The DELAYED THEORY COMBINATION (DTC) approach of *MathSAT* [5] takes this idea to the extreme: at the initialization time, introduce a proxy boolean variable e_{xy} for every equality $x = y$ between shared variables and put the definitional form $e_{xy} \Leftrightarrow x = y$ in Ψ_i for all i such that $x, y \in V_i$. Then let NODPLL communicate equalities only through their boolean proxies e_{xy}. This way we can eliminate the rules EqPropag and EqDispatch altogether.

DTC guarantees completeness even if participating theories are not convex; see case (*ii*) of Theorem 5. It is also conceptually simple and so is used in the theoretical system $DPLL(T_1, \ldots T_n)$ [3]. The disadvantage is that it requires addition of a potentially large set of new variables. *Yices* [10] adopts DTC but goes a step further and curtails proliferation of proxies e_{xy} by introducing them only "on demand", thus ignoring provably useless ones. On the other hand, in *Simplify* [9] and in *CVC Lite* [2] propagation of equalities can occur directly, without creating propositional proxy variables.

Techniques for equality propagation clearly deserve further study and experimentation; NODPLL provides a flexible medium to express various approaches.

5.7 Participating Solvers

The requirements on the deductive strength of individual solvers can be read off from Figure 5 and are exactly as specified by the $DPLL(T)$ framework [16]. 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 [16].

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 [10]. This issue has been only recently raised [12] and is awaiting proper treatment.

6 Conclusion

Gaps between published algorithms and their actual implementations are inevitable, but in the SMT area they are often too large. Clarification efforts are needed and recent work on the $DPPL(T)$ architecture [16] 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.

References

1. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
2. Barrett, C.: Checking Validity of Quantifier-free Formulas in Combinations of First-Order Theories. PhD thesis, Stanford University (2002)
3. Barrett, C., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Splitting on demand in SAT Modulo Theories. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, Springer, Heidelberg (2006)
4. Bonacina, M.P., Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Decidability and undecidability results for Nelson-Oppen and rewrite-based decision procedures. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, Springer, Heidelberg (2006)
5. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., van Rossum, P., Ranise, S., Sebastiani, R.: Efficient theory combination via boolean search. Information and Computation 204(10), 1493–1525

6. Bryant, R., Lahiri, S., Seshia, S.: Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, Springer, Heidelberg (2002)
7. Conchon, S., Krstić, S.: Strategies for combining decision procedures. Theoretical Computer Science 354(2), 187–210 (2006)
8. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
9. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: A theorem prover for program checking. Journal of the ACM 52(3), 365–473 (2005)
10. Dutertre, B., de Moura, L.: The Yices SMT solver. Technical report, SRI International (2006)
11. Eén, N., Sörensen, N.: An extensible SAT solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, Springer, Heidelberg (2004)
12. Krstić, S., Goel, A., Grundy, J., Tinelli, C.: Combined satisfiability modulo parametric theories. In: TSDM 2000. LNCS, vol. 4424, Springer, Heidelberg (2007) (extended version) available at `ftp://ftp.cs.uiowa.edu/pub/tinelli/papers/KrsGGT-RR-06.pdf`
13. Mitchell, J.C.: Foundations of Programming Languages. MIT Press, Cambridge (1996)
14. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient sat solver. In: Conference on Design Automation (DAC), ACM Press, New York (2001)
15. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Transactions on Programming Languages and Systems 1(2), 245–257 (1979)
16. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). Journal of the ACM 53(6), 937–977 (2006)
17. Ranise, S., Ringeissen, C., Zarba, C.G.: Combining data structures with nonstably infinite theories using many-sorted logic. In: Gramlich, B. (ed.) Frontiers of Combining Systems. LNCS (LNAI), vol. 3717, Springer, Heidelberg (2005)
18. Reynolds, J.C.: Types, abstraction and parametric polymorphism. In: Mason, R.E.A., (ed.) Information Processing: 9th World Computer Congress, pp. 513–523. North-Holland (1983)
19. Tinelli, C., Harandi, M.: A new correctness proof of the Nelson-Oppen combination procedure. In: Frontiers of Combining Systems (FroCoS), vol. 3 of Applied Logic, pp. 103–120 (1996)
20. Tinelli, C., Zarba, C.: Combining nonstably infinite theories. Journal of Automated Reasoning 34(3), 209–238 (2005)

A Appendix

Proof of Theorem 2

There is little here that is not contained in the proofs of Theorems 3, 4, 5. Yet, we give a full proof of Theorem 2 because it may be of independent interest and because it may serve as an introduction to the more involved proofs that follow.

Starting the proof of termination of DPLL, define the relation $M \preceq M'$ on checkpointed sequences 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 with respect to the ordering $\preceq$, 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:

If l occurs in M, then it occurs only once and $\bar{l}$ does not occur at all. (3)

It follows now that in every run

$$s_{\text{init}} \to s_1 \to s_2 \to \dots \tag{4}$$

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 $C = \mathsf{no\_cflct}$), the number of occurrences of Conflict in (4) 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 (4) 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:

Every literal in C occurs in M. (5)

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 multised ordering that the ordering of M induces on subsets of M.[5]

For the rest of the proof, assume that $s_\diamond = \langle \Psi_\diamond, M_\diamond, C_\diamond \rangle$ is a final state obtained by running DPLL initialized with the set of clauses Ψ_{init}. It remains to prove

$$C_\diamond = \mathsf{no\_cflct} \text{ or } C_\diamond = \varnothing; \tag{6}$$

$$\text{If } C_\diamond = \varnothing \text{ then } \Psi_{\text{init}} \text{ is unsatisfiable;} \tag{7}$$

$$\text{If } C_\diamond = \mathsf{no\_cflct}, \text{ then } M_\diamond \text{ is a model for } \Psi_{\text{init}}. \tag{8}$$

Proof of (6). Assuming the contrary, suppose $C_\diamond$ is a non-empty set and let l be an arbitrary element of it. By (5), l occurs in $M_\diamond$. Considering a particular run $s_{\text{init}} \to s_1 \to s_2 \ldots \to s_\diamond$, let s_n be the last state in this run such that $s_n.M$ does not contain l. Note that the transition $s_n \to s_{n+1}$ 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

$$s_{n+1}.M \text{ is a prefix of } s_i.M \text{ for all } i > n. \tag{9}$$

Indeed, if (9) is not true, then in the execution $s_{n+1} \to \cdots \to s_\diamond$ 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 $i > n$.

[5] "$M \sqsupset_{\text{mul}} N$ 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 the transition $s_n \to s_{n+1}$ is a $\mathsf{UnitPropag}$ or $\mathsf{BackJump}$, then $\mathsf{Explain}$ is applicable to s_{n+1} and so by (9), $\mathsf{Explain}$ 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 $\Box$) 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 $\mathsf{BackJump}$ or Learn applies to $s_\diamond$. In both cases we run into contradiction with the finality of $s_\diamond$.

Proof of (7). We prove by simultaneous induction that the following two properties are invariants of DPLL. Note that (7) is a special case of (11) when $k = 0$.

$$\textit{Ψ is equivalent with Ψ_{init}} \tag{10}$$

$$\textit{If } C = \{l_1, \dots, l_k\}, \textit{ then } \Psi_{\text{init}} \models \bar{l}_1 \vee \cdots \vee \bar{l}_k \tag{11}$$

Both properties are trivially true for initial states. Assuming $s \to s'$ is a transition based on a rule R of DPLL, and s satisfies (10) and (11), we proceed to prove that s' satisfies these conditions too.

If R is not Learn, then $s'.\Psi = s.\Psi$ and so s' satisfies (10) by induction hypothesis. If R is Learn, the proof that s' satisfies (10) is a simple application of the induction hypotheses (10) and (11). Thus, s' satisfies (10) in all cases.

Now we prove that s' satisfies (11). The only non-trivial cases are when R is $\mathsf{Conflict}_i$ or $\mathsf{Explain}_i$, since in all other cases we have $s'.C = s.C$. When R is $\mathsf{Conflict}_i$, from the guard $\bar{l}_1 \vee \cdots \vee \bar{l}_k \in s.\Psi$ and (10), we obtain the desired relation $\Psi_{\text{init}} \models \bar{l}_1 \vee \cdots \vee \bar{l}_k$ immediately.

Suppose R is $\mathsf{Explain}_i$ 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 $\mathsf{Explain}$ and (10) imply $\Psi_{\text{init}} \models \bar{l}_1 \vee \cdots \vee \bar{l}_k \vee l$. The required relation $\Psi_{\text{init}} \models \gamma \vee \bar{l}_1 \vee \cdots \vee \bar{l}_k$ follows immediately.

Proof of (8). From (5) 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 (10), we only need to prove $M_\diamond \models \gamma$ for every clause $\gamma = l_1 \vee \cdots \vee l_k$ in $\Psi_\diamond$. But if this were not true, we would have $\bar{l}_i \in M_\diamond$ for $i = 1, \dots, k$ and so $\mathsf{Conflict}$ would be applicable to the stat $s_\diamond$, contradicting its finality.

Proof of Theorems 3 and 4

Assuming that the input formula Φ is fixed, we will write s_{init} for s^{Φ}_{init}. 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:

$$\textit{All local stacks contain the same number of occurrences of } \Box; \tag{12}$$

$$\textit{In any local stack, any literal can occur at most once;} \tag{13}$$

$$\textit{If l occurs in $M_{\mathbb{B}}$, then $\bar{l}$ does not occur in $M_{\mathbb{B}}$.} \tag{14}$$

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

Consider an arbitrary execution sequence (finite or infinite)

$$\pi: \; s_{\mathrm{init}} = s_0 \to s_1 \to s_2 \to \dots \tag{15}$$

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, $s_{\mathrm{init}}.O$ is the empty sequence. Then, if the transition $s_k \to s_{k+1}$ is based on the rule R, let $s_{k+1}.O = s_k.O$ if R is Learn, $\mathsf{Conflict}_i$ or $\mathsf{Explain}_i$; if R is $\mathsf{BackJump}$, 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:

$R:$	Decide	Infer_i	$\mathsf{LitDispatch}_i$	$\mathsf{EqDispatch}_i$	$\mathsf{LitPropag}_i$	$\mathsf{EqPropag}_i$
$U:$	$\square + \langle p^\epsilon, \mathbb{B}, \mathbb{B}\rangle$	$\langle l, i, i\rangle$	$\langle p^\epsilon, j, i\rangle$	$\langle x = y, j, i\rangle$	$\langle p^\epsilon, i, \mathbb{B}\rangle$	$\langle x = y, i, \mathbb{E}\rangle$

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 1. *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. □

Corollary 1. *If the rule* $\mathsf{LitDispatch}_i$ *or* $\mathsf{EqDispatch}_i$ *applies to a reachable state, then $j \neq i$. (The number j here is from the rule as stated in Figure 5.)*

Proof. Let s be a state in which $\mathsf{LitDispatch}_i$ applies. (We omit the discussion of the entirely analogous $\mathsf{EqDispatch}$ 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 1, $\langle p^\epsilon, j, j\rangle \in s.O$. In particular $p^\epsilon \in M_j$, which together with the guard $p^\epsilon \notin M_i$ implies $i \neq j$. □

For each state s, define

$$s.C^* = \begin{cases} \mathsf{no\_cflct} & \text{if } s.C = \mathsf{no\_cflct} \\ \{\langle l, i, i\rangle \mid \langle l, i\rangle \in s.C\} & \text{otherwise} \end{cases}$$

Lemma 2. *(a) For every reachable state s with $s.C \neq$* $\mathsf{no\_cflct}$*, all elements of $s.C^*$ occur in $s.O$.*
(b) If s is a reachable state and $s \to s'$ is a transition based on the rule $\mathsf{Explain}_i$*, then $s'.C^* \sqsubset_{\mathrm{mul}} s.C^*$, where $\sqsubseteq_{\mathrm{mul}}$ is the multiset ordering*[6] *induced by the relation $\sqsubseteq$ on the set of triples occurring in $s'.O = s.O$.*

[6] The guard of $\mathsf{Explain}_i$ implies that neither $s.C^*$ nor $s'.C^*$ can be $\mathsf{no\_cflct}$, so they can be compared by $\sqsubseteq_{\mathrm{mul}}$.

Proof. First we prove *(a)* by induction. Let s, s' be two consecutive states in any execution sequence π and let the transition $s \to s'$ be based on a rule R. If R is any of the first six rules in Figure 5, or if R is Learn, then $s'.C^* = s.C^*$ and $s'.O$ contains $s.O$. If R is $\mathsf{BackJump}$, then $s'.C \neq \mathsf{no\_cflct}$. In all these cases, there is nothing to check. Finally, if R is either $\mathsf{Conflict}_i$ or $\mathsf{Explain}_i$, 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 1*(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

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

where $\langle l_1, i_1 \rangle, \ldots, \langle l_k, i_k \rangle \prec_{s.M_i} \langle l, i \rangle$. Thus, by Lemma 1*(a,b)*,

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

for $\nu = 1, \ldots, k$. □

Proof of Theorem 3. Part 1: Termination. Immediately from the definition of the global stack, for every NODPLL transition $s \to s'$ we have $s.O = s'.O$ if the transition is based on Learn, $\mathsf{Conflict}$, or $\mathsf{Explain}$, 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 the proof of Theorem 2.

Assume now the theorem is not true and suppose the execution sequence π in (15) is infinite. From the previous paragraphs, we have

$$s_{\mathrm{init}}.O \preceq s_1.O \preceq s_2.O \preceq \ldots \tag{16}$$

The invariant (13) (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 (16) can be strict.

Since (16) is thus an eventually constant sequence, the first six rules of NODPLL and $\mathsf{BackJump}$ (which all necessarily change $s.O$) can occur only finitely many times in π. Since $\mathsf{Conflict}$ applies only when $C = \mathsf{no\_cflct}$ and replaces it with a set, and since only $\mathsf{BackJump}$ can make $C = \mathsf{no\_cflct}$ again, we derive that $\mathsf{Conflict}$ 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 $\mathsf{Explain}$.

Now, for some m, we have that all transitions in the subsequence $s_m \to s_{m+1} \to \ldots$ of π are based on $\mathsf{Explain}$. By Lemma 2, this yields an infinite descending chain $s_m.C^* \sqsupset_{\mathrm{mul}} s_{m+1}.C^* \sqsupset_{\mathrm{mul}} \ldots$, contradicting well-foundedness of the multiset ordering. This finishes the proof that NODPLL is terminating. □

Recall that 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 $M_{\mathsf{dec}}^{[n]}$ be the sequence of the first n decision literals. We will also need the notation for conflict clauses: if $s.C = \{l_1, \ldots, l_k\}$, define $s.C^{\mathsf{cls}} = \bar{l}_1 \vee \cdots \vee \bar{l}_k$, and if $s.C = \mathsf{no\_cflct}$, define $s.C^{\mathsf{cls}} = \mathsf{true}$. Note that $s.C^{\mathsf{cls}} = \mathsf{false}$ if $s.C = \varnothing$.

Lemma 3. *The following are invariants of* NODPLL*:*

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

Proof. The initial state obviously satisfies all three conditions.

We prove (a) and (b) by simultaneous induction. If $s \to s'$ 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 $s'.\Psi = s.\Psi \cup \{s.C^{\mathsf{cls}}\}$, 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): $\Psi_{\text{init}} \models_{\mathcal{T}} s'.C^{\mathsf{cls}}$. The only non-trivial cases are when R is $\mathsf{Conflict}_i$ or $\mathsf{Explain}_i$, since in all other cases we have either $s'.C = s.C$ or $s'.C = \mathsf{true}$ (the latter happens if R is $\mathsf{BackJump}$). When R is $\mathsf{Conflict}_i$, the truth of the guard $s.\Psi_i, l_i, \ldots, l_k \models_i \mathsf{false}$ directly implies $s.\Psi_i \models_i s'.C^{\mathsf{cls}}$. By induction hypothesis (a) $\Psi_{\text{init}} \models_{\mathcal{T}} s.\Psi_i$. Combining the two facts proves our goal.

Suppose finally R is $\mathsf{Explain}_i$. The clause $s'.C^{\mathsf{cls}}$ is obtained from $s.C^{\mathsf{cls}}$ by removing the literal $\bar{l}$ and adding literals $\bar{l}_1, \ldots, \bar{l}_k$. We have $s.\Psi_i, l_i, \ldots, l_k \models_i l$ from the guard of our rule. We also have $\Psi_{\text{init}} \models_{\mathcal{T}} s.C^{\mathsf{cls}}$ 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.

$$\text{\textit{For every } } i \text{ \textit{and} } n,\ \Psi_{\text{init}}, M_{\mathsf{dec}}^{[n]} \models_{\mathcal{T}} M_i^{[n]}. \tag{17}$$

Again, we reason by induction. If R, the rule used in a transition $s \to s'$, is one of the first six (from Decide to $\mathsf{EqPropag}$) in Figure 5, it is easy to see by direct examination that s' satisfies (17) 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 $s \to s'$ is based on $\mathsf{BackJump}$. The only fact needed to prove that s' satisfies (17) that is not immediately implied by the induction hypothesis is $\Psi_{\text{init}}, M_{\mathsf{dec}}^{[m]} \models_{\mathcal{T}} \bar{l}$, where m and l are as in the rule $\mathsf{BackJump}$ in Figure 5. This follows from two facts: (1) $\Psi_{\text{init}} \models_{\mathcal{T}} s.C^{\mathsf{cls}}$, and (2) $\Psi_{\text{init}}, M_{\mathsf{dec}}^{[m]} \models_{\mathcal{T}} l'$ 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 $\mathsf{BackJump}$ implies that $l' \in s.M_{\mathbb{B}}^{[m]}$ for every $l' \in s.C - \{l\}$. □

Proof of Theorem 3. Part 2: Final States. Suppose s is a final state with $s.C \neq \mathsf{no\_cflct}$ and let $s_{\text{init}} \to s_1 \to s_2 \to \ldots \to s$ be a run leading to s. Arguing by contradiction, assume $s.C \neq \varnothing$. 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 $\mathsf{BackJump}$ 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 2 (a), $\langle l, i\rangle$ occurs in $s.M_j$ for some j, and then by Lemma 1 (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 $s_n \to s_{n+1}$. By definition of the global stack, we have $s_{n+1}.O = s_n.O^{[m]} + \langle l, i, i\rangle$ (when R is $\mathsf{BackJump}$), $s_{n+1}.O = s_n.O{+}\Box{+}\langle l, i, i\rangle$ (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 $\mathsf{BackJump}$.

Note that in all cases $s_{n+1}.O$ must be a prefix of $s.O$, because otherwise there would have been a $\mathsf{BackJump}$ to a level smaller than the level of $\langle l, i, i\rangle$ in the execution $s_{n+1} \to \cdots \to s$, 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 $\Psi_i, l_1, \ldots, l_k \models_i l$ for some $l_1, \ldots, l_k \in s_n.M_i$, 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 $\mathsf{Explain}_i$ is enabled at s—a contradiction because s is final.

Suppose now R is $\mathsf{BackJump}$: $s_{n+1}.O = s_n.O^{[m]} + \langle l, i, i\rangle$. We have $i = \mathbb{B}$ and $\langle l, i\rangle = \langle \bar{l}_0, \mathbb{B}\rangle$, where $s_n.C = \{\langle l_0, i_0\rangle, \langle l_1, i_1\rangle, \ldots, \langle l_k, i_k\rangle\}$ and $\mathsf{level}\ l_0 > k \geq \mathsf{level}\ l_\nu$ $(\nu = 1, \ldots k)$. The levels here are computed with respect to s_n. Thus, $l_1, \ldots, l_k \in s_n.M_{\mathbb{B}}$. Moreover, since the backjumping level in the transition $s_n \to s_{n+1}$ is m, from the guard of $\mathsf{BackJump}$ we have that the literals $l_1, \ldots, l_k$ 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 $\mathsf{BackJump}$ implies also that the clause $\bar{l}_0 \vee \bar{l}_1 \vee \cdots \vee \bar{l}_k$ 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 $\mathsf{Explain}_{\mathbb{B}}$ applies to s, which (again) is not possible because s is final. $\Box$

Proof of Theorem 4. This is a special case of Lemma 3(b), since $C^{\mathsf{cls}} = \mathsf{false}$ when $C = \varnothing$. $\Box$

Proof of Theorem 5

By Theorem 1, to prove that Φ is $\mathcal{T}$-satisfiable, it suffices to find an assignment M to variables occurring in $\Phi_{\mathbb{B}}$ and an arrangement Δ of other shared variables in Φ such that

$$M \models \Phi_{\mathbb{B}} \tag{18}$$

$$(\Phi_i \cup \Delta)^{\mathrm{pure}} \cup \Phi_i^{\mathrm{card}} \cup M \ \textit{is } \mathcal{T}_i\textit{-satisfiable for every } i \in \{\mathbb{E}, 1, \ldots, n\} \tag{19}$$

Since $\Psi_i = \Phi_i^{\mathrm{pure}} \cup \Phi_i^{\mathrm{card}}$, the condition (19) can be restated as

$$\Psi_i \cup \Delta \cup M \ \textit{is } \mathcal{T}_i\textit{-satisfiable for every } i \in \{\mathbb{E}, 1, \ldots, n\}. \tag{20}$$

Now, suppose we are in a final state s of NODPLL. We will first see how s determines M and $\varDelta$ such that (18) holds. Then we will show that (20) holds if we assume either of the conditions $(i), (ii)$.

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

For the rest of the proof, observe that rules of NODPLL do not modify local constraints other than $\varPsi_{\mathbb{B}}$; thus, $s.\varPsi_i = \varPsi_i$ holds for every $i \neq \mathbb{B}$.

Recall that $\varPhi_{\mathbb{E}}$ consists of formulas of the form $p \Leftrightarrow (x = y)$, where $p \in V_{\mathbb{B}}$ and x, y are shared variables. Since $M = s.M_{\mathbb{B}}$ is an assignment to $V_{\mathbb{B}}$, we have either $p \in M$, or $\bar{p} \in M$. In the first case, we must have $p \in s.M_{\mathbb{E}}$, and in the second case, we must have $\bar{p} \in s.M_{\mathbb{E}}$—otherwise, $\mathsf{LitDispatch}_{\mathbb{E}}$ would apply to s. (Note that $p, \bar{p} \in L_{\mathbb{E}}$.) Since $\varPsi_{\mathbb{E}} \cup s.M_{\mathbb{E}}$ is consistent (otherwise, $\mathsf{Conflict}_{\mathbb{E}}$ would apply to s), it follows that the set $\varPsi_{\mathbb{E}} \cup s.M_{\mathbb{E}} \cup M$ is consistent too. Consider an arbitrary model of this set and define $\varDelta$ to be the arrangement of $V_{\mathbb{E}}$ determined by that model. Clearly, (20) holds for $i = \mathbb{E}$.

Arguing by contradiction, assume (20) is not true for some i. This implies that $\varPsi_i \cup \varDelta_i \cup M$ is not $\mathcal{T}_i$-satisfiable, where $\varDelta_i$ is the subset of $\varDelta$ containing the (dis)equalities between variables in V_i. Indeed, if one has a $\mathcal{T}_i$-model for $\varPsi_i \cup \varDelta_i \cup M$, then this model extends to a model of $\varPsi_i \cup \varDelta \cup M$ by interpreting the non-V_i variables of $\varDelta$ as arbitrary elements constrained only by equalities and disequalities of $\varDelta - \varDelta_i$. (It is easy to argue that such extensions exist.)

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

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

Case 1: Assume (i) *holds.* Write $\varDelta_i = \varDelta^+ \cup \varDelta^-$, where $\varDelta^+$ and $\varDelta^-$ contain equalities and disequalities respectively. Then $\varPsi_i, M_i, \varDelta^+ \models_i \neg\varDelta^-$. For each definitional form $p \Leftrightarrow \phi$ in $\varPsi_i$, either p or $\bar{p}$ occurs in M_i (because $p \in V_{\mathbb{B}} \cap V_i$ and M is an assignment to all variables in $V_{\mathbb{B}}$). Obtain $\varPsi_i'$ from $\varPsi_i$ by replacing $p \Leftrightarrow \phi$ with ϕ or $\neg\phi$, depending on whether M_i contains p or $\neg p$. Clearly, $\varPsi_i, M_i \models_i \varPsi_i'$ and $\varPsi_i', \varDelta^+ \models_i \neg\varDelta^-$. Now, $\varPsi_i' \cup \varDelta^+$ is a set of $\mathcal{T}_i$-literals (here we use the assumption that $\varPsi_i$ contains no cardinality constraints) and $\varDelta^-$ is a disjunction of equalities. Since $\mathcal{T}_i$ is convex, we must have $\varPsi_i' \cup \varDelta^+ \models_i x = y$ for some $x, y \in V_i$ such that $(x \neq y) \in \varDelta^-$. In other words, $\varPsi_i, M_i \models_i x = y$ 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 $\mathsf{EqPropag}_i$ does not apply, we must have $(x = y) \in s.M_{\mathbb{E}}$. This contradicts consistency of $s.M_{\mathbb{E}} \cup \varDelta$ that has already been established, finishing 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 $p \in s.M_{\mathbb{E}}$, or $\bar{p} \in M_i$ and $\bar{p} \in s.M_{\mathbb{E}}$. 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 $\Psi_i, M_i \models_i \mathsf{false}$ and therefore applicability of $\mathsf{Conflict}_i$ 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 $\mathsf{Conflict}_i$ and it would go through even if the rules Infer_i $(i = 1, \ldots, n)$ 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 $i \in \{1, \ldots, n\}$, one of the following conditions holds:

(i') $\mathcal{T}_i$ is convex and $\Phi_i^{\text{card}} = \varnothing$;
(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 with minimal changes.

Finally, we give an example showing that in case (i) the completeness does not necessarily hold without the assumption $\Phi_i^{\text{card}} = \varnothing$.

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 the two-element set $\iota(\alpha)$.

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 $j \in \{1, \ldots, 5\} \smallsetminus \{i\}$. 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 $\Phi_1, \ldots, \Phi_5$ 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.