Delayed theory combination vs. Nelson-Oppen for satisfiability modulo theories: a comparative analysis

**Roberto Bruttomesso · Alessandro Cimatti ·
Anders Franzen · Alberto Griggio · Roberto Sebastiani**

Abstract Most state-of-the-art approaches for Satisfiability Modulo Theories $(SMT(\mathcal{T}))$ rely on the integration between a SAT solver and a decision procedure for sets of literals in the background theory $\mathcal{T}$ ($\mathcal{T}$-solver). Often $\mathcal{T}$ is the combination $\mathcal{T}_1 \cup \mathcal{T}_2$ of two (or more) simpler theories $(SMT(\mathcal{T}_1 \cup \mathcal{T}_2))$, s.t. the specific $\mathcal{T}_i$-solvers must be combined. Up to a few years ago, the standard approach to $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$ was to integrate the SAT solver with one combined $\mathcal{T}_1 \cup \mathcal{T}_2$-solver, obtained from two distinct $\mathcal{T}_i$-solvers by means of evolutions of *Nelson and Oppen's (NO)* combination procedure, in which the $\mathcal{T}_i$-solvers deduce and exchange interface equalities.

R. Bruttomesso (✉)
Università della Svizzera Italiana, Lugano, Switzerland
e-mail: roberto.bruttomesso@unisi.ch

A. Cimatti · A. Franzen
FBK-Irst, Povo, Trento, Italy

A. Cimatti
e-mail: cimatti@fbk.eu

A. Franzen
e-mail: franzen@fbk.eu

A. Griggio · R. Sebastiani
DISI, University of Trento, Povo, Trento, Italy

A. Griggio
e-mail: griggio@disi.unitn.it

R. Sebastiani
e-mail: rseba@disi.unitn.it

 Springer

Nowadays many state-of-the-art SMT solvers use evolutions of a more recent $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$ procedure called *Delayed Theory Combination (DTC)*, in which each $\mathcal{T}_i$-*solver* interacts directly and only with the SAT solver, in such a way that part or all of the (possibly very expensive) reasoning effort on interface equalities is delegated to the SAT solver itself. In this paper we present a comparative analysis of DTC vs. NO for $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$. On the one hand, we explain the advantages of DTC in exploiting the power of modern SAT solvers to reduce the search. On the other hand, we show that the extra amount of Boolean search required to the SAT solver can be controlled. In fact, we prove two novel theoretical results, for both convex and non-convex theories and for different deduction capabilities of the $\mathcal{T}_i$-solvers, which relate the amount of extra Boolean search required to the SAT solver by DTC with the number of deductions and case-splits required to the $\mathcal{T}_i$-*solvers* by NO in order to perform the same tasks: (i) under the same hypotheses of deduction capabilities of the $\mathcal{T}_i$-*solvers* required by NO, DTC causes no extra Boolean search; (ii) using $\mathcal{T}_i$-*solvers* with limited or no deduction capabilities, the extra Boolean search required can be reduced down to a negligible amount by controlling the quality of the $\mathcal{T}$-conflict sets returned by the $\mathcal{T}$-*solvers*.

1 Introduction

Satisfiability Modulo a Theory $\mathcal{T}$ ($SMT(\mathcal{T})$) is the problem of checking the satisfiability of a quantifier-free (or ground) first-order formula with respect to a given first-order theory $\mathcal{T}$. (Actually, some SMT solvers can also handle quantifiers to some extent.) Theories of interest for many applications are, e.g., the theory $\mathcal{EUF}$ of equality and uninterpreted functions, the theory of difference constraints $\mathcal{DL}$ (over the rationals $\mathcal{DL}(\mathbb{Q})$ or over the integers $\mathcal{DL}(\mathbb{Z})$), the quantifier-free fragment of Linear Arithmetic over the rationals $\mathcal{LA}(\mathbb{Q})$ and that over the integers $\mathcal{LA}(\mathbb{Z})$, the theory of arrays $\mathcal{AR}$ and the theory of bit-vectors $\mathcal{BV}$.

The prominent "lazy" approach to $SMT(\mathcal{T})$ which underlies most state-of-the-art systems (e.g., BARCELOGIC [32], CVC3 [4], DPT [27], MATHSAT [10], YICES [19], Z3 [15]) is based on extensions of SAT technology: a SAT engine, typically based on modern implementations of the DPLL algorithm [45, 46], is modified to enumerate Boolean assignments, and integrated with a decision procedure for sets of literals in the theory $\mathcal{T}$ ($\mathcal{T}$-*solver*).

In many practical applications of SMT, the theory $\mathcal{T}$ is a combination of two (or more) theories $\mathcal{T}_1$ and $\mathcal{T}_2$, $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$. For better readability, in this paper we always deal with only two theories $\mathcal{T}_1$ and $\mathcal{T}_2$; the discourse generalizes to more than two theories. For instance, an atom of the form $f(x + 4y) = g(2x - y)$, that combines uninterpreted function symbols (from $\mathcal{EUF}$) with arithmetic functions (from $\mathcal{LA}(\mathbb{Z})$), could be used to naturally model in a uniform setting the abstraction of some functional blocks in an arithmetic circuit (see e.g. [7, 11]).

The work on combining decision procedures (i.e., $\mathcal{T}$-solvers in our terminology) for distinct theories was pioneered by Nelson and Oppen [30, 33] and Shostak [39].[1] In particular, Nelson and Oppen established the theoretical foundations onto which most current combined procedures are still based on (hereafter *Nelson-Oppen (NO) logical framework*). They also proposed a general-purpose procedure for integrating $\mathcal{T}_i$-*solvers* into one combined $\mathcal{T}$-*solver* (hereafter *Nelson-Oppen (NO) procedure*), based on the deduction and exchange of (disjunctions of) equalities between shared variables (*interface equalities*).

Up to a few years ago, the standard approach to $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$ was thus to integrate the SAT solver with one combined $\mathcal{T}_1 \cup \mathcal{T}_2$-solver, obtained from two distinct $\mathcal{T}_i$-solvers by means of the NO combination procedure. Variants and improvements of the NO procedure were implemented in the CVC/CVCLite [2], ICS [16], Simplify [17], Verifun [22], Zapato [1] lazy SMT tools. In particular, [5] introduced two important improvements of N.O procedure: they show that purificaton is not necessary, because it is possible to use equalities between shared terms as interface equalities, and they show how to use Shostak's canonizers [39] to achieve e_{ij}-deduction.

More recently Bozzano et al. [9, 10] proposed *Delayed Theory Combination, DTC*, a novel combination procedure in which each $\mathcal{T}_i$-*solver* interacts directly and only with the SAT solver, in such a way that part or all of the (possibly very expensive) reasoning effort on interface equalities is delegated to the SAT solver itself. Variants and improvements of the DTC procedure are currently implemented in the CVC3 [3, 4], DPT [27],[2] MathSAT [10], Yices [19], and Z3 [15] lazy SMT tools; in particular, Yices [19], and Z3 [15] introduced many important improvements on the DTC schema (e.g., that of generating interface equalities on-demand, and important "model-based" heuristic to drive the Boolean search on the interface equalities); CVC3 [4] combines the main ideas from DTC with that of splitting-on-demand [3], which pushes even further the idea of delegating to the DPLL engine part of the reasoning effort previously due to the $\mathcal{T}_i$-*solvers*.

In this paper we present a detailed comparative analysis of DTC wrt. NO procedure for $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$.

On the one hand, we analyze, compare and discuss the behavior of the NO and the DTC procedures for $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$, both with convex and with non-convex theories, and we highlight some important advantages of DTC in exploiting the power of modern lazy DPLL-based SAT solvers: first, it allows for learning in form of clauses the results of reasoning on interface equalities, so that to reuse them in future branches; second, it does not require exhaustive deduction capabilities to the $\mathcal{T}_i$-*solvers*, although it can fully exploit them; third, it nicely encompasses the case of non-convex theories.

On the other hand, we prove and discuss two novel results, for both convex and non-convex theories and for different deduction capabilities of the $\mathcal{T}_i$-*solvers*, which

---

[1]Nowadays there seems to be a general consensus on the fact that Shostak's procedure should not be considered as an independent combination method, rather as a collection of ideas on how to implement Nelson-Oppen's combination method efficiently [5, 17, 35].

[2]Notice that, although [27] speak of "Nelson-Oppen with DPLL", their formalism implements and further improves the key ideas of DTC: Boolean Reasoning also on interface equalities, conflict clauses involving interface equalities, deduction of interface equalities exploited as $\mathcal{T}$-propagation. (See Section 5)

relate the amount of extra Boolean search required to the SAT solver by DTC with the number of deductions and case-splits required to the $\mathcal{T}_i$-*solvers* by NO in order to perform the same tasks. We show that, by exploiting the full power of advanced SMT techniques like Early Pruning, $\mathcal{T}$-propagation, $\mathcal{T}$-backjumping and $\mathcal{T}$-learning, DTC can be implemented in such a way as to mimic the behavior of NO, so that:

(i)   under the same hypotheses of e_{ij}-deduction capabilities of the $\mathcal{T}_i$-*solvers* required by NO, DTC requires no extra Boolean search;

(ii) using $\mathcal{T}_i$-*solvers* with limited or no e_{ij}-deduction capabilities, the extra Boolean search required can be reduced down to a negligible amount by controlling the quality of the $\mathcal{T}$-conflict sets returned by the $\mathcal{T}$-*solvers*.

Content of the paper   The paper is structured as follows. In Section 2 we provide the main logical background necessary for the comprehension of the paper, plus we recall the Nelson-Oppen logical framework. In Section 3 we describe and discuss the schema and the main features of a modern lazy SMT solver. In Section 4 we describe the NO procedure and discuss issues related to its integration within the lazy SMT schema. In Section 5 we describe the DTC procedure and discuss its advantages wrt. the NO procedure. In Section 6 we prove and discuss the two theoretical results mentioned above. Finally, in Section 7 we draw some conclusions.

A shorter version of this paper has been presented at LPAR'06 conference [12].

2 Logical background

In this section we recall the main logical background necessary for the comprehension of the paper, plus we introduce the notation, conventions and terminology adopted. In particular, we recall the Nelson-Oppen logical framework, which provides the logical foundations for both the Nelson-Oppen and the Delayed Theory Combination procedures.

2.1 Basic definitions and notation

We assume the usual syntax and semantics of first-order logic, and the usual first-order notions of interpretation, satisfiability, validity, logical consequence, and theory, as given, e.g., in [20]. In this paper we restrict our attention to *quantifier-free* formulae on some theory $\mathcal{T}$.[3] All the theories $\mathcal{T}$ we consider are first-order theories with equality, which means that the equality symbol $=$ is a predefined predicate and it is always interpreted as the identity on the underlying domain. Consequently, $=$ is interpreted as a relation which is reflexive, symmetric, transitive, and it is also a congruence.

Notationally, we will often use the prefix "$\mathcal{T}$-" to denote "in the theory $\mathcal{T}$": e.g., we call a "$\mathcal{T}$-formula" a formula in (the signature of) $\mathcal{T}$, "$\mathcal{T}$-model" a model in $\mathcal{T}$, and so on. We call a *theory solver* for $\mathcal{T}$ ($\mathcal{T}$-*solver*) a procedure establishing whether any

---

[3]Notice that in SMT the variables are implicitly existentially quantified, and hence equivalent to Skolem constants. To this extent, as it is common practice in the SMT community, we often call "variables" uninterpreted constants and "Boolean variables" 0-ary uninterpreted predicates.

given finite conjunction of quantifier-free $\mathcal{T}$-literals (or equivalently, any given finite set of $\mathcal{T}$-literals) is $\mathcal{T}$-satisfiable or not. Given a $\mathcal{T}$-inconsistent set of $\mathcal{T}$-literals $\mu = \{l_1, \ldots, l_n\}$, a $\mathcal{T}$-*conflict set* η is a $\mathcal{T}$-inconsistent subset of μ. A literal l is *redundant* in $\mathcal{T}$-conflict set η iff it plays no role in the $\mathcal{T}$-unsatisfiability of η, i.e., $\eta \setminus \{l\} \models_{\mathcal{T}} \bot$; η is a *minimal* if it contains no redundant literals.

We use the Greek letters φ, ψ to represent $\mathcal{T}$-formulas, the capital letters A_i's and B_i's to represent Boolean atoms, and the Greek letters α, β, γ to represent $\mathcal{T}$-atoms in general, the letters l_i's to represent $\mathcal{T}$-literals, the letters μ, η to represent sets of $\mathcal{T}$-literals. If l is a negative $\mathcal{T}$-literal $\neg\beta$, then by "$\neg l$" we conventionally mean β rather than $\neg\neg\beta$. We sometimes represent a set of literals as the conjunction of its components (e.g., by $\neg\mu$ me mean $\neg(\bigwedge_{l_i \in \mu} l_i)$ or even the clause $\bigvee_{l_i \in \mu} \neg l_i$). We sometimes write a clause in the form of an implication: $\bigwedge_i l_i \rightarrow \bigvee_j l_j$ for $\bigvee_i \neg l_i \vee \bigvee_j l_j$ and $\bigwedge_i l_i \rightarrow \bot$ for $\bigvee_i \neg l_i$.

We define the following functions. The function $Atoms(\varphi)$ takes a $\mathcal{T}$-formula φ and returns the set of distinct atomic formulas (atoms) occurring in the $\mathcal{T}$-formula φ. The bijective function $\mathcal{T}2\mathcal{B}$ ("Theory-to-Boolean") and its inverse $\mathcal{B}2\mathcal{T} \stackrel{\text{def}}{=} \mathcal{T}2\mathcal{B}^{-1}$ ("Boolean-to-Theory") are s.t. $\mathcal{T}2\mathcal{B}$ maps Boolean atoms into themselves and non-Boolean $\mathcal{T}$-atoms into fresh Boolean atoms—so that two atom instances in φ are mapped into the same Boolean atom iff they are syntactically identical—and extend to $\mathcal{T}$-formulas and sets of $\mathcal{T}$-formulas in the obvious way—i.e., $\mathcal{B}2\mathcal{T}(\neg\varphi_1) \stackrel{\text{def}}{=} \neg\mathcal{B}2\mathcal{T}(\varphi_1)$, $\mathcal{B}2\mathcal{T}(\varphi_1 \bowtie \varphi_2) \stackrel{\text{def}}{=} \mathcal{B}2\mathcal{T}(\varphi_1) \bowtie \mathcal{B}2\mathcal{T}(\varphi_2)$ for each Boolean connective $\bowtie$, $\mathcal{B}2\mathcal{T}(\{\varphi_i\}_i) \stackrel{\text{def}}{=} \{\mathcal{B}2\mathcal{T}(\varphi_i)\}_i$. $\mathcal{T}2\mathcal{B}$ and $\mathcal{B}2\mathcal{T}$ are also called *Boolean abstraction* and *Boolean refinement* respectively. To this extent, we frequently use the superscript p to denote Boolean abstractions: given a $\mathcal{T}$-expression e, we write e^p to denote $\mathcal{T}2\mathcal{B}(e)$, and vice versa. If $\mathcal{T}2\mathcal{B}(\mu) \models \mathcal{T}2\mathcal{B}(\varphi)$, then we say that μ *propositionally satisfies* φ, written $\mu \models_p \varphi$.

2.2 The Nelson-Oppen logical framework

A theory $\mathcal{T}$ is *stably-infinite* iff every quantifier-free $\mathcal{T}$-satisfiable formula is satisfiable in an infinite model of $\mathcal{T}$. Notice that $\mathcal{EUF}$, $\mathcal{DL}(\mathbb{Q})$, $\mathcal{DL}(\mathbb{Z})$, $\mathcal{LA}(\mathbb{Q})$, $\mathcal{LA}(\mathbb{Z})$ are stably-infinite, whereas e.g. theories of fixed-width bit-vectors $\mathcal{BV}$ are not. (E.g., the theory of bit-vectors of width n admits only models of cardinality up to 2^n and thus it is not stably-infinite.)

A theory $\mathcal{T}$ is *convex* iff, for every collection $l_1, \ldots, l_k, e, e'$ of literals in $\mathcal{T}$ s.t. e, e' are in the form $(x = y)$, x, y being variables, we have that

$$\{l_1, \ldots, l_k\} \models_{\mathcal{T}} (e \vee e') \iff \{l_1, \ldots, l_k\} \models_{\mathcal{T}} e \ or \ \{l_1, \ldots, l_k\} \models_{\mathcal{T}} e'.$$

Notice that $\mathcal{EUF}$, $\mathcal{DL}(\mathbb{Q})$, $\mathcal{LA}(\mathbb{Q})$ are convex, whereas $\mathcal{DL}(\mathbb{Z})$ and $\mathcal{LA}(\mathbb{Z})$ are not. In fact, e.g.:

$$\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \models_{\mathcal{LA}(\mathbb{Z})} ((v = v_0) \vee (v = v_1)),$$
$$\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \not\models_{\mathcal{LA}(\mathbb{Z})} (v = v_0),$$
$$\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \not\models_{\mathcal{LA}(\mathbb{Z})} (v = v_1).$$

Notice also that every convex theory whose models are non-trivial (i.e., s.t. the domains of the models have all cardinality strictly greater than one) is stably-infinite [5].

Consider two theories $\mathcal{T}_1$, $\mathcal{T}_2$ with equality and disjoint signatures Σ_1, Σ_2. A $\Sigma_1 \cup \Sigma_2$-term t is an *i-term* iff either it is a variable or it has the form $f(t_1, ..., t_n)$, where f is in Σ_i. Notice that a variable is both a 1-term and a 2-term. A non-variable subterm s of an *i*-term t is *alien* if s is a *j*-term, and all superterms of s in t are *i*-terms, where $i, j \in \{1, 2\}$ and $i \neq j$. An *i*-term is *i-pure* if it does not contain alien subterms. An atom (or a literal) is *i-pure* if it contains only *i*-pure terms and its predicate symbol is either equality or in Σ_i. A $\mathcal{T}_1 \cup \mathcal{T}_2$-formula φ is said to be *pure* if every atom occurring in the formula is *i*-pure for some $i \in \{1, 2\}$. (Intuitively, φ is pure if each atom can can be seen as belonging to one theory $\mathcal{T}_i$ only.) Every non-pure $\mathcal{T}_1 \cup \mathcal{T}_2$ formula φ can be converted into an equi-satisfiable pure formula φ' by recursively labeling each alien subterm t with a fresh variable v_t, and by adding the atom $(v_t = t)$. E.g.:

$$(f(x + 3y) = g(2x - y))$$
$$\implies (f(v_{x+3y}) = g(v_{2x-y})) \wedge (v_{x+3y} = x + 3y) \wedge (v_{2x-y} = 2x - y).$$

This process is called *purification*, and is linear in the size of the input formula. Thus, henceforth we assume w.l.o.g. that all input formulas $\varphi \in \mathcal{T}_1 \cup \mathcal{T}_2$ are pure.[4]

If φ is a pure $\mathcal{T}_1 \cup \mathcal{T}_2$ formula, then v is an *interface variable* for φ iff it occurs in both 1-pure and 2-pure atoms of φ. An equality $(v_i = v_j)$ is an *interface equality* for φ iff v_i, v_j are interface variables for φ. We assume a unique representation for $(v_i = v_j)$ and $(v_j = v_i)$. Henceforth we denote the interface equality $(v_i = v_j)$ by "e_{ij}"; to this extent, we also say that a $\mathcal{T}$-conflict set η is $\neg e_{ij}$-*minimal* if it contains no redundant negated interface equality $\neg e_{ij}$; a $\mathcal{T}$-solver is called $\neg e_{ij}$-*minimal* if it always returns e_{ij}-minimal conflict sets.

Consider two decidable stably-infinite theories with equality $\mathcal{T}_1$ and $\mathcal{T}_2$ and disjoint signatures Σ_1 and Σ_2 (often called *Nelson-Oppen theories*) and consider a pure conjunction of $\mathcal{T}_1 \cup \mathcal{T}_2$-literals $\mu \stackrel{\text{def}}{=} \mu_{\mathcal{T}_1} \wedge \mu_{\mathcal{T}_2}$ s.t. $\mu_{\mathcal{T}_i}$ is i-pure for each i. Nelson and Oppen's key observation is that μ is $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable if and only if it is possible to find two satisfying interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ s.t. $\mathcal{I}_1 \models_{\mathcal{T}_1} \mu_{\mathcal{T}_1}$ and $\mathcal{I}_2 \models_{\mathcal{T}_2} \mu_{\mathcal{T}_2}$ which agree on all equalities on the shared variables. This is stated in the following theorem.[5]

Theorem 1 [40] *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two stably-infinite theories with equality and disjoint signatures; let $\mu \stackrel{\text{def}}{=} \mu_{\mathcal{T}_1} \wedge \mu_{\mathcal{T}_2}$ be a conjunction of $\mathcal{T}_1 \cup \mathcal{T}_2$-literals s.t. $\mu_{\mathcal{T}_i}$ is i-pure for each i. Then $\mu_{\mathcal{T}_1} \wedge \mu_{\mathcal{T}_2}$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable if and only if there exists some equivalence relation $e(.,.)$ over $Vars(\mu_{\mathcal{T}_1}) \cap Vars(\mu_{\mathcal{T}_2})$ s.t. $\mu_{\mathcal{T}_i} \wedge \mu_e$ is $\mathcal{T}_i$-satisfiable for every i, where:*

$$\mu_e \stackrel{\text{def}}{=} \bigwedge_{(v_i, v_j) \in e(.,.)} (v_i = v_j) \wedge \bigwedge_{(v_i, v_j) \notin e(.,.)} \neg(v_i = v_j). \tag{1}$$

μ_e *is called the* arrangement *of $e(.,.)$.*

---

[4]Notice that this assumption is made only for the sake of better comprehension of the paper, because both NO procedure and the DTC procedure can work also with non-pure formulas, thanks to some techniques described in [5].

[5]Since [30] many different formulations of NO correctness and completeness results have been presented (e.g. [30, 33, 40, 41]). Here we adopt a notational variant of that in [40].

Example 1 Consider the following pure conjunction of $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Z})$-literals $\mu \stackrel{\text{def}}{=} \mu_{\mathcal{EUF}} \wedge \mu_{\mathcal{LA}(\mathbb{Z})}$ s.t.

$$\mu_{\mathcal{EUF}} : \neg(f(v_1) = f(v_2)) \wedge \neg(f(v_2) = f(v_4)) \wedge (f(v_3) = v_5) \wedge (f(v_1) = v_6)$$
$$\mu_{\mathcal{LA}(\mathbb{Z})} : (v_1 \geq 0) \wedge (v_1 \leq 1) \wedge (v_5 = v_4 - 1) \wedge (v_3 = 0) \wedge (v_4 = 1) \tag{2}$$
$$\wedge (v_2 \geq v_6) \wedge (v_2 \leq v_6 + 1).$$

Here $v_1, \ldots, v_6$ are interface variables, because they occur in both $\mathcal{EUF}$ and $\mathcal{LA}(\mathbb{Q})$-pure terms. We consider the arrangement

$$\mu_e \stackrel{\text{def}}{=} (v_1 = v_4) \wedge (v_3 = v_5) \wedge \bigwedge_{(v_i = v_j) \notin \{(v_1 = v_4), (v_3 = v_5)\}} \neg(v_i = v_j).$$

It is easy to see that $\mu_{\mathcal{EUF}} \wedge \mu_e$ is $\mathcal{EUF}$-consistent (because no equality or congruence constraint is violated) and that $\mu_{\mathcal{LA}(\mathbb{Z})} \wedge \mu_e$ is $\mathcal{LA}(\mathbb{Z})$-consistent (e.g., $v_3 = v_5 = 0$, $v_1 = v_4 = 1$, $v_2 = 4$, $v_6 = 3$ is a $\mathcal{LA}(\mathbb{Z})$-model). Thus, by Theorem 1, μ is $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Z})$-consistent.

Overall, Nelson-Oppen results reduce the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability problem of a set of pure literals μ to that of finding (the arrangement of) an equivalence relation on the shared variables which is consistent with both pure parts of μ. The condition of having only pure conjunctions as input allows to partition the problem into two independent $\mathcal{T}_i$-satisfiability problems $\mu_{\mathcal{T}_i} \wedge \mu_e$, whose $\mathcal{T}_i$-satisfiability can be checked separately. The condition of having stably-infinite theories is sufficient to guarantee enough values in the domain to allow the satisfiability of every possible set of disequalities one may encounter.

A significant research effort has been paid to extend NO framework by releasing the conditions it is based on (purity the of inputs, stably-infiniteness and signature-disjointness of the theories.) We briefly overview some of them.[6]

First, [5] shows that the purity condition is not really necessary in practice. Intuitively, one may consider alien terms as if they were variables, and consider equalities between alien terms as interface equalities. We refer the reader to [5] for details.

Many approaches have been presented in order to release the condition of stably-infiniteness (e.g., [6, 23, 34, 42–44]). In particular, [42, 44] proposed a method which extends the NO framework by reasoning not only on interface equalities, but also on particular cardinality constraints; this method has been extended to many-sorted logics in [34]; the problem has been further explored theoretically in [6], and related to that of combining rewrite-based decision procedures. Finally, the paradigm in [42] has been recently extended in [28] so that to handle also parametric theories.

A few approaches have been proposed also to release the condition of signature-disjointness [25, 41]. A theoretical framework addressing this problem was proposed in [41], which allowed for producing semi-decision procedures. [25] proposed a general theoretical framework based on classical model theory. An even more general framework for combining decision procedures, which captures that in [25] as a subcase, has been recently presented in [26].

---

[6]The list of references and approaches listed here is by no means intended to be exhaustive.

All these results, however, involve a theoretical analysis which exceeds the scope of this paper, so that we refer the reader to the cited bibliography for further details.

3 Modern SMT solvers

In order to facilitate the comprehension of the rest of the paper, in this section we need to explain with some detail the schema and the main features of a modern "lazy" SMT solver based on the DPLL algorithm. (See [36] for a much more detailed explanation.)

3.1 The lazy SMT schema and its main optimizations

In a nutshell a lazy SMT solver works as follows. Given an input $\mathcal{T}$-formula φ, a SAT solver is used to enumerate a complete set of truth assignments μ_i^p satisfying the Boolean abstraction $\varphi^p \overset{\text{def}}{=} \mathcal{T}2\mathcal{B}(\varphi)$. The set of literals $\mu_i \overset{\text{def}}{=} \mathcal{B}2\mathcal{T}(\mu_i^p)$ is then fed to a $\mathcal{T}$-*solver*, which checks the satisfiability in $\mathcal{T}$ of μ_i. The process is repeated until either a $\mathcal{T}$-satisfiable μ_i is found, so that φ is $\mathcal{T}$-satisfiable, or no more truth assignments μ_i^p can found, so that φ is $\mathcal{T}$-unsatisfiable. This approach is referred to as "lazy", in contraposition to the "eager" approach, consisting in encoding the formula into an equi-satisfiable SAT formula (when possible) and into feeding it to a SAT solver.

Figure 1 represents the schema of a modern lazy SMT solver based on a DPLL engine (see e.g. [46]). The input φ and μ are a $\mathcal{T}$-formula and a reference to an (initially empty) set of $\mathcal{T}$-literals respectively. The DPLL solver embedded in

```
1.      SatValue T-DPLL(T-formula φ, T-assignment & μ) {
2.          if (T-preprocess(φ,μ) == Conflict)
3.              return unsat;
4.          φᵖ= T2B(φ); μᵖ= T2B(μ);
5.          while (1) {
6.              T-decide_next_branch(φᵖ,μᵖ);
7.              while (1) {
8.                  status = T-deduce(φᵖ,μᵖ);
9.                  if (status == sat) {
10.                     μ = B2T(μᵖ);
11.                     return sat; }
12.                 else if (status == Conflict) {
13.                     blevel = T-analyze_conflict(φᵖ,μᵖ);
14.                     if (blevel == 0)
15.                         return unsat;
16.                     else T-backtrack(blevel,φᵖ,μᵖ);
17.                 }
18.                 else break;
19.     } } }
```

Fig. 1 An online schema of $\mathcal{T}$-DPLL based on modern DPLL

$\mathcal{T}$-DPLL reasons on and updates φ^p and μ^p, and $\mathcal{T}$-DPLL maintains some data structure encoding the bijective mapping $\mathcal{T}2\mathcal{B}/\mathcal{B}2\mathcal{T}$ on atoms.[7]

$\mathcal{T}$-`preprocess` simplifies φ into a simpler formula, and updates μ if it is the case, so that to preserve the $\mathcal{T}$-satisfiability of $\varphi \wedge \mu$. If this process produces some conflict, then $\mathcal{T}$-DPLL returns unsat. $\mathcal{T}$-`preprocess` may combine most or all the Boolean preprocessing steps available from SAT literature with theory-dependent rewriting steps on the $\mathcal{T}$-literals of φ. This step involves also the conversion to CNF of the input formula, if required.

$\mathcal{T}$-`decide_next_branch` selects some literal l^p and adds it to μ^p. It plays the same role as the standard literal selection heuristic `decide_next_branch` in DPLL [46], but it may take into consideration also the semantics in $\mathcal{T}$ of the literals to select. (This operation is called *decision*, l^p is called *decision literal* and the number of decision literals in μ after this operation is called the *decision level* of l^p.)

$\mathcal{T}$-`deduce`, in its simplest version, behaves similarly to `deduce` in DPLL [46]: it iteratively deduces Boolean literals l^p which derive propositionally from the current assignment (i.e., s.t. $\varphi^p \wedge \mu^p \models l^p$) and updates φ^p and μ^p accordingly. (The iterative application of unit-propagation performed by `deduce` and $\mathcal{T}$-`deduce` is often called *Boolean Constraint Propagation, BCP*.) This step is repeated until one of the following facts happens:

(i) μ^p propositionally violates φ^p ($\mu^p \wedge \varphi^p \models \bot$). If so, $\mathcal{T}$-`deduce` behaves like `deduce` in DPLL, returning Conflict.
(ii) μ^p satisfies φ^p ($\mu^p \models \varphi^p$). If so, $\mathcal{T}$-`deduce` invokes $\mathcal{T}$-*solver* on $\mathcal{B}2\mathcal{T}(\mu^p)$: if $\mathcal{T}$-*solver* returns sat, then $\mathcal{T}$-`deduce` returns sat; otherwise, $\mathcal{T}$-`deduce` returns Conflict.
(iii) no more literals can be deduced. If so, $\mathcal{T}$-`deduce` returns Unknown.

A slightly more elaborated version of $\mathcal{T}$-`deduce` can invoke $\mathcal{T}$-*solver* on $\mathcal{B}2\mathcal{T}(\mu^p)$ also if μ^p does not yet satisfy φ^p: if $\mathcal{T}$-*solver* returns unsat, then $\mathcal{T}$-`deduce` returns Conflict. (This enhancement is called *Early Pruning, EP*.)

Moreover, during EP calls, if $\mathcal{T}$-*solver* is able to perform deductions in the form $\eta \models_{\mathcal{T}} l$ s.t. $\eta \subseteq \mu$ and $l^p \stackrel{\text{def}}{=} \mathcal{T}2\mathcal{B}(l)$ is an unassigned literal in φ^p, then $\mathcal{T}$-`deduce` can append l^p to μ^p and propagate it. (This enhancement is called $\mathcal{T}$-*propagation*.)

$\mathcal{T}$-`analyze_conflict` is an extension of *analyze_conflict* of DPLL [46]: if the conflict produced by $\mathcal{T}$-`deduce` is caused by a Boolean failure (case (i) above), then $\mathcal{T}$-`analyze_conflict` produces a Boolean conflict set η^p and the corresponding value *blevel* of the decision level where to backtrack; if instead the conflict is caused by a $\mathcal{T}$-inconsistency revealed by $\mathcal{T}$-*solver*, then $\mathcal{T}$-`analyze_conflict` produces the Boolean abstraction $\eta^p \stackrel{\text{def}}{=} \mathcal{T}2\mathcal{B}(\eta)$ of the $\mathcal{T}$-conflict set η produced by $\mathcal{T}$-*solver*.

---

[7]Hereafter we implicitly assume that all functions called in $\mathcal{T}$-DPLL have direct access to $\mathcal{T}2\mathcal{B}/\mathcal{B}2\mathcal{T}$, and that both $\mathcal{T}2\mathcal{B}$ and $\mathcal{B}2\mathcal{T}$ require constant time for mapping each atom.

$\mathcal{T}$-backtrack behaves analogously to *backtrack* in DPLL [46]: once the conflict set η^p and *blevel* have been computed, it adds the clause $\neg\eta^p$ to φ^p, either temporarily and permanently, and backtracks up to *blevel*. (These features are called $\mathcal{T}$-*learning* and $\mathcal{T}$-*backjumping*.)

Example 2 Consider the following $\mathcal{LA}(\mathbb{Q})$-formula φ and its Boolean abstraction φ^p:

$$
\begin{aligned}
&c_1 : \varphi = \{\neg(2x_2 - x_3 > 2) \vee A_1\} && \varphi^p = \{\neg B_1 \vee A_1\} \\
&c_2 : \quad\quad \{\neg A_2 \vee (x_1 - x_5 \le 1)\} && \{\neg A_2 \vee B_2\} \\
&c_3 : \quad\quad \{(3x_1 - 2x_2 \le 3) \vee A_2\} && \{B_3 \vee A_2\} \\
&c_4 : \quad\quad \{\neg(2x_3 + x_4 \ge 5) \vee \neg(3x_1 - x_3 \le 6) \vee \neg A_1\} && \{\neg B_4 \vee \neg B_5 \vee \neg A_1\} \\
&c_5 : \quad\quad \{A_1 \vee (3x_1 - 2x_2 \le 3)\} && \{A_1 \vee B_3\} \\
&c_6 : \quad\quad \{(x_2 - x_4 \le 6) \vee (x_5 = 5 - 3x_4) \vee \neg A_1\} && \{B_6 \vee B_7 \vee \neg A_1\} \\
&c_7 : \quad\quad \{A_1 \vee (x_3 = 3x_5 + 4) \vee A_2\} && \{A_1 \vee B_8 \vee A_2\}
\end{aligned}
$$

Consider the Boolean search tree in Fig 2a. Suppose $\mathcal{T}$-decide_next_branch selects, in order, $\neg B_5$, B_8, B_6 (occurring in c_4, c_7 and c_6 respectively). In this process $\mathcal{T}$-deduce cannot unit-propagate any literal and EP calls to $\mathcal{T}$-*solver* produce no information.

Then $\mathcal{T}$-decide_next_branch selects $\neg B_1$ (occurring in c_1). By EP, $\mathcal{T}$-deduce invokes $\mathcal{T}$-*solver* on $\mathcal{B}2\mathcal{T}(\{\neg B_5, B_8, B_6, \neg B_1\})$:

$$\{\neg(3x_1 - x_3 \le 6), (x_3 = 3x_5 + 4), (x_2 - x_4 \le 6), \neg(2x_2 - x_3 > 2)\}.$$

$\mathcal{T}$-*solver* not only returns sat, but also it performs the deduction

$$\{\neg(3x_1 - x_3 \le 6), \neg(2x_2 - x_3 > 2)\} \models_{\mathcal{LA}(\mathbb{Q})} \neg(3x_1 - 2x_2 \le 3) \tag{3}$$

of the literal $\neg(3x_1 - 2x_2 \le 3)$ occurring in c_3 and c_5. The corresponding Boolean literal $\neg B_3$ is added to μ^p and propagated ($\mathcal{T}$-propagation). Hence A_1, A_2 and B_2 are unit-propagated from c_5, c_3 and c_2.

$c_8 : B_5 \vee \neg B_8 \vee \neg B_2$

$c_9 : B_5 \vee B_1 \vee \neg B_3$
$c_8 : B_5 \vee \neg B_8 \vee \neg B_2$

$c_8 : B_5 \vee \neg B_8 \vee \neg B_2$
$c_8' : B_5 \vee \neg B_8 \vee B_1$

(a) Example 2   (b) Example 3   (c) Example 4
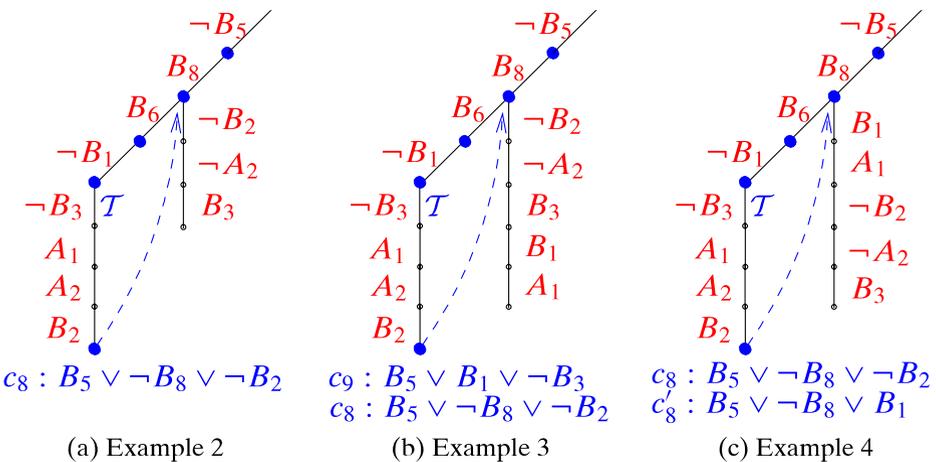
Fig. 2 Boolean search sub-trees in the scenarios of Examples 2, 3 and 4 respectively (**a–c**). (A *diagonal line*, a *vertical line* and a *vertical line* tagged with "$\mathcal{T}$" denote literal selection, unit propagation and $\mathcal{T}$-propagation respectively; a *bullet* denotes a call to $\mathcal{T}$-*solver*)

Then $\mathcal{T}$-deduce invokes $\mathcal{T}$-*solver* on $\mathcal{B}2\mathcal{T}$ ($\{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\}$):

$$\{\neg(3x_1 - x_3 \le 6), (x_3 = 3x_5 + 4), (x_2 - x_4 \le 6),$$
$$\neg(2x_2 - x_3 > 2), \neg(3x_1 - 2x_2 \le 3), (x_1 - x_5 \le 1)\}$$

which is inconsistent because of the 1st, 2nd, and 6th literals. Thus, $\mathcal{T}$-*solver* returns unsat and the conflict clause

$$c_8 \stackrel{\text{def}}{=} B_5 \vee \neg B_8 \vee \neg B_2$$

corresponding to the conflict. Then $\mathcal{T}$-DPLL adds c_8 (either temporarily or permanently) to the clause set and backtracks, popping from μ^p all literals up to $\{\neg B_5, B_8\}$, and then unit-propagates $\neg B_2$ on c_8 ($\mathcal{T}$-backjumping and $\mathcal{T}$-learning). Then, starting from $\{\neg B_5, B_8, \neg B_2\}$, also $\neg A_2$ and B_3 are unit-propagated on c_2 and c_3 respectively, and the search proceeds from there.

An important further improvement of $\mathcal{T}$-deduce is the following: when $\mathcal{T}$-*solver* is invoked on EP calls and performs a deduction $\eta \models_{\mathcal{T}} l$ (step (iii) above), then the clause $\mathcal{T}2\mathcal{B}(\neg \eta \vee l)$ (called *deduction clause*) can be added to φ^p, either temporarily or permanently. The deduction clause will be used for the future Boolean search, with benefits analogous to those of $\mathcal{T}$-learning. To this extent, notice that $\mathcal{T}$-propagation can be seen as a unit-propagation on a deduction clause. (As both $\mathcal{T}$-conflict clauses and deduction clauses are $\mathcal{T}$-valid, they are also called $\mathcal{T}$-*lemmas*.)

Example 3 Consider the formulas φ and φ^p of Example 2 and the search tree of Fig. 2b. The deduction step (3) can be represented as

$$(3x_1 - x_3 \le 6) \vee (2x_2 - x_3 > 2) \vee \neg(3x_1 - 2x_2 \le 3),$$

corresponding to the deduction clause:

$$c_9 \stackrel{\text{def}}{=} B_5 \vee B_1 \vee \neg B_3,$$

which is returned to $\mathcal{T}$-DPLL, which adds it (either temporarily or permanently) to the clause set. If this is the case, then also B_1 and hence A_1 are unit-propagated on c_9 and c_1 respectively.

Another important improvement of $\mathcal{T}$-analyze_conflict and $\mathcal{T}$-backtrack [24] is that of building from $\neg \eta^p$ also a "mixed Boolean+theory conflict clause", by recursively removing non-decision literals l^p from the clause $\neg \eta^p$ (in this case called *conflicting clause*) by resolving the latter with the clause C_{l^p} which caused the unit-propagation of l^p (called the *antecedent clause* of l^p); if l^p was propagated by $\mathcal{T}$-propagation, then the deduction clause is used as antecedent clause. This is done until the conflict clause contains no non-decision literal which has been assigned after the last decision (*last-UIP strategy*) or at most one such non-decision literal (*first-UIP strategy*).[8]

---

[8]These are standard techniques for SAT solvers to build the Boolean conflict clauses [45].

Example 4 Consider again the formulas φ and φ^p of Examples 1 and 2 and the search tree of Fig. 2c. $\mathcal{T}$-analyze_conflict may also look for a mixed Boolean+theory conflict clause c_8' by resolving backward c_8 with c_2 and c_3, (i.e., with the antecedent clauses of B_2 and A_2) and with the deduction clause c_9 (which "caused" the propagation of $\neg B_3$):

$$\dfrac{\overbrace{B_5 \vee \neg B_8 \vee \neg B_2}^{c_8:\ \text{\textit{theory conflicting clause}}} \quad \overbrace{\neg A_2 \vee B_2}^{c_2}}{\dfrac{B_5 \vee \neg B_8 \vee \neg A_2}{\dfrac{B_5 \vee \neg B_8 \vee B_3}{\underbrace{B_5 \vee \neg B_8 \vee B_1}_{c_8':\ \text{\textit{mixed Boolean+theory conflict clause}}}}(A_2)}(B_2)} \ \ \overbrace{B_3 \vee A_2}^{c_3} \ \ \overbrace{B_5 \vee B_1 \vee \neg B_3}^{c_9}(\neg B_3)$$

finding the mixed Boolean+theory conflict clause c_8' : $B_5 \vee \neg B_8 \vee B_1$. (Notice that, unlike with c_8 and c_9, $\mathcal{B2T}(c_8') = (3x_1 - x_3 \leq 6) \vee \neg(x_3 = 3x_5 + 4) \vee (2x_2 - x_3 > 2)$ is not $\mathcal{LA}(\mathbb{Q})$-valid.) This pattern corresponds to the last-UIP schema in [45], because the process terminates when all literals which have been propagated after the last decision (B_2, A_2, $\neg B_3$) have been removed from the conflict clause.

Then (Fig. 2c) $\mathcal{T}$-backtrack pops from μ^p all literals up to $\{\neg B_5, B_8\}$, and then unit-propagates B_1 on c_8', so that also A_1 is unit-propagated on c_1. If also c_8 is added to the clause set, then also B_1, $\neg A_2$ and B_3 are unit-propagated on c_8, c_2 and c_3 respectively.

On the whole, $\mathcal{T}$-DPLL differs from the DPLL schema of [46] because it exploits:

- an extended notion of deduction and propagation of literals: not only unit propagation ($\mu^p \wedge \varphi^p \models l^p$), but also $\mathcal{T}$-propagation ($\mathcal{B2T}(\mu^p) \models_\mathcal{T} \mathcal{B2T}(l^p)$);
- an extended notion of *conflict*: not only *Boolean conflict* ($\mu^p \wedge \varphi^p \models \bot$), but also *theory conflict* ($\mathcal{B2T}(\mu^p) \models_\mathcal{T} \bot$), or even *mixed Boolean+theory conflict* ($\mathcal{B2T}(\mu^p \wedge \varphi^p) \models_\mathcal{T} \bot$).

$\mathcal{T}$-DPLL is a coarse abstraction of the algorithms underlying most state-of-the art SMT solvers, including BARCELOGIC, CVC3, DPT, MATHSAT, YICES, Z3. Many other optimizations and improvements have been proposed in the literature, which are not of interest for this paper. (We refer the reader, e.g., to [36] for a survey.)

3.2 Discussion

In order to fully understand the differences between the N.O. and DTC procedures, it is important to discuss the role played by Early Pruning, $\mathcal{T}$-propagation, $\mathcal{T}$-backjumping and $\mathcal{T}$-learning in a lazy SMT solver like $\mathcal{T}$-DPLL, and their strict relation with important features of the $\mathcal{T}$-*solvers*. We elaborate a little on these issues.

Early pruning (EP) is based on the observation that, if the $\mathcal{T}$-unsatisfiability of an assignment μ is detected during its construction, then this prevents checking the $\mathcal{T}$-satisfiability of all the up to $2^{|Atoms(\varphi)| - |\mu|}$ total truth assignments which extend μ. In general, early pruning may introduce a very relevant reduction of the Boolean search space, and consequently of the number of calls to $\mathcal{T}$-*solvers*. This may partly

be counterbalanced by the fact that EP may cause useless calls to $\mathcal{T}$-*solver*. Different strategies for interleaving EP calls and DPLL steps have been presented in the literature.

$\mathcal{T}$-**propagation**   allows $\mathcal{T}$-*solver* for efficiently driving the Boolean search of DPLL and to prune a priori branches corresponding to $\mathcal{T}$-inconsistent sets of literals. For theories when the deduction is cheap, this may bring to dramatic performance improvements [14, 31, 32]. In general, there is a tradeoff between the benefits of search pruning and the expense of $\mathcal{T}$-propagation, so that different strategies for interleaving $\mathcal{T}$-propagation and DPLL steps have been presented in the literature.

$\mathcal{T}$-**learning**   implements the intuitive idea "never repeat the same mistake twice" as with standard DPLL: once the clause $\neg \eta^p$ is added in conjunction to φ^p, $\mathcal{T}$-DPLL will never again generate any branch containing η^p, because as soon as $|\eta^p| - 1$ literals in η^p are assigned to true, the remaining literal will be immediately assigned to false by unit-propagation on $\neg \eta^p$. The $\mathcal{T}$-conflict set returned by the $\mathcal{T}$-*solver* drives the future search of DPLL to avoid generating the same $\mathcal{T}$-conflict set again.

$\mathcal{T}$-**backjumping**   implements the intuitive idea "go back to the earliest point where you could have made a smarter assignment if only you had known the conflict clause in advance, and make it". The $\mathcal{T}$-conflict set returned by the $\mathcal{T}$-*solver* drives DPLL to jump up to many different decision levels. In particular, e.g., in case of a mixed Boolean+theory conflict clause obtained from the $\mathcal{T}$-conflict by the last UIP strategy, $\mathcal{T}$-analyze_conflict reveals the most recent decision which caused deterministically the $\mathcal{T}$-conflict (alone or by any combination of unit- and $\mathcal{T}$-propagation) and backtracks to the earliest point where DPLL can take the opposite decision in accordance to the conflict clause.

The effectiveness of these techniques is strongly related to some important features of the $\mathcal{T}$-solvers.

Incrementality and backtrackability   Due to EP calls, it is often the case that $\mathcal{T}$-*solver* is invoked sequentially on *incremental* assignments, in a stack-based manner, like in the following trace (left column first, then right) [8]:

$\mathcal{T}$-*solver* (μ_1)	$\Longrightarrow$ sat	Undo μ_4, μ_3, μ_2	
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2)$	$\Longrightarrow$ sat	$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2')$	$\Longrightarrow$ sat
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Longrightarrow$ sat	$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2' \cup \mu_3')$	$\Longrightarrow$ sat
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Longrightarrow$ unsat	...	

Thus, a key efficiency issue of $\mathcal{T}$-*solver* is that of being *incremental* and *backtrackable*. *Incremental* means that $\mathcal{T}$-*solver* "remembers" its computation status from one call to the other, so that, whenever it is given in input an assignment $\mu_1 \cup \mu_2$ such that μ_1 has just been proved $\mathcal{T}$-satisfiable, it avoids restarting the computation from scratch by restarting the computation from the previous status. *Backtrackable* means that it is possible to undo steps and return to a previous status on the stack in an efficient manner.[9]

---

[9]"Backtrackable" is also called "*resettable*" by Nelson and Oppen [30].

There are incremental and backtrackable versions of the congruence closure algorithm for $\mathcal{EUF}$ [17, 31], of the Bellman-Ford algorithm for $\mathcal{DL}$ [14, 32], and of the Simplex LP procedure for $\mathcal{LA}(\mathbb{Q})$ [18].

Deduction of unassigned literals ($\mathcal{T}$-deduction) For many theories it is possible to implement $\mathcal{T}$-solver so that, when returning sat, it can also perform efficiently enough a set of deductions in the form $\eta \models_{\mathcal{T}} l$, s.t. $\eta \subseteq \mu$ and l is a literal on a not-yet-assigned atom in φ.[10] We say that $\mathcal{T}$-solver is *deduction-complete* if it can perform all possible such deductions, or say that no such deduction can be performed.

For $\mathcal{EUF}$, the computation of congruence closure allows for efficiently deducing positive equalities [31]; for $\mathcal{DL}$, a very efficient implementation of a deduction-complete $\mathcal{T}$-solver was presented by [14, 32]; for $\mathcal{LA}$ the task is much harder, and only $\mathcal{T}$-solvers capable of incomplete forms of deduction have been presented [18].

Generation of "high-quality" $\mathcal{T}$-lemmas A key efficiency issue is the "quality" of the $\mathcal{T}$-lemmas returned by $\mathcal{T}$-solver: the less redundant literals it contains, the more effective it will be for $\mathcal{T}$-backjumping (which may allow for higher jumps) and for $\mathcal{T}$-learning (which may prune more branches in the future).

There exist conflict-set-producing variants for the Bellman-Ford algorithm for $\mathcal{DL}$, [14], for the Simplex LP procedures for $\mathcal{LA}(\mathbb{Q})$ [18] and for the congruence closure algorithm for $\mathcal{EUF}$ [31], producing high-quality $\mathcal{T}$-conflict sets.

Another feature is essential for implementing Nelson-Oppen procedure (see later).

Deduction of interface equalities (e_{ij}-deduction) For most theories it is possible to implement $\mathcal{T}$-solver so that , when returning sat, it can also perform a set of deductions in the form $\mu \models_{\mathcal{T}} e$ (if $\mathcal{T}$ is convex) or in the form $\mu \models_{\mathcal{T}} \bigvee_j e_j$ (if $\mathcal{T}$ is not convex) s.t. $e, e_1, ..., e_n$ are equalities between variables occurring in μ. (Notice that here the deduced equalities need not occur in the input formula φ.) As typically $e, e_1, ..., e_n$ are interface equalities, we call these forms of deductions e_{ij}-*deductions*, and we say that a $\mathcal{T}$-solver is e_{ij}-*deduction-complete* if it can perform all possible such deductions, or say that no such deduction can be performed.

e_{ij}-deduction-complete $\mathcal{T}$-solvers are often (implicitly) implemented by means of *canonizers* [39]. Intuitively, a *canonizer* canon$_{\mathcal{T}}$ for a theory $\mathcal{T}$ is a function which maps a term t into another term canon$_{\mathcal{T}}(t)$ in *canonical* form, that is, canon$_{\mathcal{T}}$ maps terms which are semantically equivalent in $\mathcal{T}$ into the same term. Thus, if x_{t_1}, x_{t_2} are interface variables labeling the terms t_1 and t_2 respectively, then the interface equality $(x_{t_1} = x_{t_2})$ can be deduced in $\mathcal{T}$ if and only if canon$_{\mathcal{T}}(t_1)$ and canon$_{\mathcal{T}}(t_2)$ are syntactically identical.

It is important to highlight that, whilst for some theories $\mathcal{T}$ like $\mathcal{EUF}$ e_{ij}-deduction-completeness can be cheap [31], for some other theories it can be extremely expensive, often much more expensive than $\mathcal{T}$-satisfiability itself. (E.g., for $\mathcal{DL}(\mathbb{Z})$

---

[10]Notice that, in principle, every $\mathcal{T}$-solver has $\mathcal{T}$-deduction capabilities, as it is always possible to call $\mathcal{T}$-solver$(\mu \cup \{\neg l\})$ for every unassigned literal l. We call this technique, *plunging* [17]. In practice plunging is very inefficient.

the problem is NP-complete [29] even though solving a set of difference constraints requires only quadratic time in worst case.)

4 SMT for combined theories via Nelson-Oppen's procedure

In [30] Nelson and Oppen (and later Shostak [39]) proposed also a general-purpose combination procedure for combining two (or more) $\mathcal{T}_i$-solvers into one $\mathcal{T}_1 \cup \mathcal{T}_2$-solver if all $\mathcal{T}_i$'s are Nelson-Oppen theories, which is based on the logical framework of Section 2.2. The combined $\mathcal{T}_1 \cup \mathcal{T}_2$-solver works by performing a structured interchange of interface equalities (disjunctions of interface equalities if $\mathcal{T}_i$ is non-convex) which are inferred by either $\mathcal{T}_i$-solver and then propagated to the other, until convergence is reached.

In order to leverage the procedure to a $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$ context, the combined $\mathcal{T}_1 \cup \mathcal{T}_2$-solver is then integrated with DPLL according to the lazy SMT schema described in Section 3.1.

4.1 The Nelson-Oppen procedure

A basic architectural schema of $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$ via N.O. is described in Fig. 3. (Here we provide only a high-level description; the reader may refer, e.g., to [5, 17, 21, 30, 37, 38] for more details.) We assume that all $\mathcal{T}_i$'s are N.O. theories and their $\mathcal{T}_i$-solvers are e_{ij}-deduction complete (see Section 3.2).

We consider first the case in which both theories are convex. The combined $\mathcal{T}_1 \cup \mathcal{T}_2$-solver receives from DPLL a pure set of literals μ, and partitions it into $\mu_{\mathcal{T}_1} \cup \mu_{\mathcal{T}_2}$, s.t. $\mu_{\mathcal{T}_i}$ is i-pure, and feeds each $\mu_{\mathcal{T}_i}$ to the respective $\mathcal{T}_i$-solver. Each $\mathcal{T}_i$-solver, in turn:

  (i)   checks the $\mathcal{T}_i$-satisfiability of $\mu_{\mathcal{T}_i}$,
  (ii) deduces all the interface equalities deriving from $\mu_{\mathcal{T}_i}$,
 (iii) passes them to the other $\mathcal{T}$-solver, which adds it to his own set of literals.

Fig. 3 A basic architectural schema of $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$ via the N.O. procedure
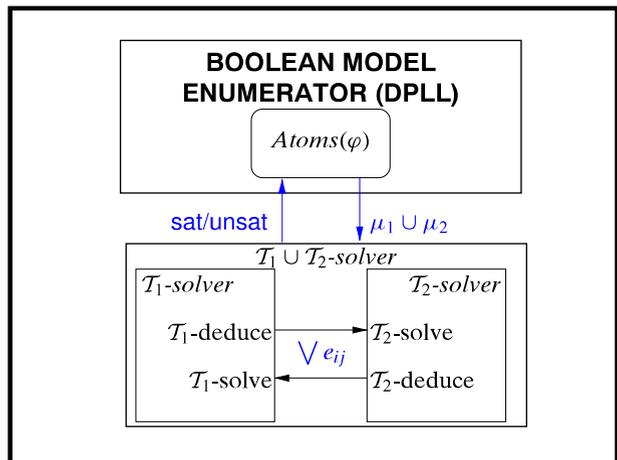
This process is repeated until either one $\mathcal{T}_i$-*solver* detects inconsistency ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsatisfiable), or no more e_{ij}-deduction is possible ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable).

Example 5 Consider the following pure $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$-formula φ:

$$
\begin{aligned}
\mathcal{EUF} : \quad & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\
\mathcal{LA}(\mathbb{Q}) : \quad & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\
\textit{Both} : \quad & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).
\end{aligned} \tag{4}
$$

$v_0, v_1, v_2, v_3, v_4, v_5$ are interface variables, v_6, v_7, v_8 are not. (Thus, e.g., $(v_0 = v_1)$ is an interface equality, whilst $(v_0 = v_6)$ is not.) $RESET_5$ is a Boolean variable.

Consider the search tree in Fig. 4. After the first run of unit-propagations, assume DPLL selects the literal $RESET_5$, resulting in the assignment $\mu \overset{\text{def}}{=} \mu_{\mathcal{EUF}} \cup \mu_{\mathcal{LA}(\mathbb{Q})}$ s.t.

$$
\begin{aligned}
\mu_{\mathcal{EUF}} = \ & \{(v_3 = h(v_0)), (v_4 = h(v_1)), (v_6 = f(v_2)), (v_7 = f(v_5)), \neg(v_6 = v_7)\} \\
\mu_{\mathcal{LA}(\mathbb{Q})} = \ & \{(v_0 \leq v_1), (v_0 \geq v_1), (v_2 = v_3 - v_4), (v_5 = 0)\},
\end{aligned} \tag{5}
$$

which propositionally satisfies φ. Now, the set of literals $\mu_{\mathcal{EUF}}$ is given to the $\mathcal{EUF}$-solver, which reports its consistency and deduces no new interface equality. Then the set $\mu_{\mathcal{LA}(\mathbb{Q})}$ is given to the $\mathcal{LA}(\mathbb{Q})$-solver, which reports consistency and deduces the interface equality $v_0 = v_1$, which is passed to the $\mathcal{EUF}$-solver. The new set $\mu_{\mathcal{EUF}} \cup \{(v_0 = v_1)\}$ is still $\mathcal{EUF}$-consistent, but this time the $\mathcal{EUF}$-solver can deduce from it the equality $(v_3 = v_4)$, which is in turn passed to the $\mathcal{LA}(\mathbb{Q})$-solver, which deduces $(v_2 = v_5)$. The $\mathcal{EUF}$-solver is then invoked again to check the $\mathcal{EUF}$-consistency of the assignment $\mu_{\mathcal{EUF}} \cup \{(v_0 = v_1), (v_2 = v_5)\}$: since this check fails, the Nelson-Oppen procedure reports the $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$-unsatisfiability of the whole assignment μ.

In the case in which at least one theory is non-convex, the N.O. procedure becomes more complicated, because the two solvers need to exchange arbitrary disjunctions
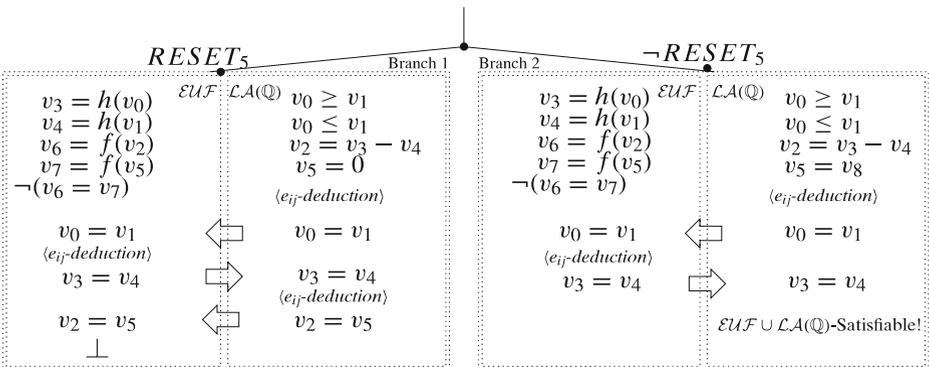
Fig. 4 Search tree for the scenario of Examples 5 (branch 1) and 7 (branch 2)

of interface equalities. As each $\mathcal{T}_i$-*solver* can handle only conjunctions of literals, the disjunctions must be managed by means of case splitting and of backtrack search. Thus, in order to check the consistency of a set of literals, the combined $\mathcal{T}_1 \cup \mathcal{T}_2$-*solver* must internally explore a number of branches which depends on how many disjunctions of equalities are exchanged at each step: if the current set of literals is μ, and one of the $\mathcal{T}_i$-*solver* sends the disjunction $\bigvee_{k=1}^{n}(e_{ij})_k$ to the other, the latter must further investigate up to n branches to check the consistency of each of the $\mu \cup \{(e_{ij})_k\}$ sets separately.

4.2 Discussion

N.O. procedure was originally conceived to combine decision procedures on *sets of literals*, much before the lazy SMT approach was conceived, so that it was not tailored for interfacing with a SAT solver and for exploiting its full power. In what follows we analyze, with the help of a few examples, the behaviour of N.O. procedure when used within a lazy SMT context, with both convex and non-convex theories.

4.2.1 Returning "e_{ij}-unaware" conflict clauses

First, in the above schema the DPLL solver is not made aware of the interface equalities e_{ij}, so that the latter cannot occur in conflict clauses. Therefore, in order to construct the $\mathcal{T}_1 \cup \mathcal{T}_2$-conflict clause, it is necessary to resolve backwards the last conflict clause with (the deduction clauses corresponding to) the e_{ij}-deductions performed by each $\mathcal{T}_i$-*solver*. This causes the generation of possibly very long and lowly-informative conflict(ing) clauses.

Example 6 Consider the scenario of Example 5 (left branch in Fig. 4). Starting from the final $\mathcal{EUF}$ conflict, and resolving backwards wrt. the deductions performed, it is possible to obtain a final $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$-conflict clause as follows:

$\mathcal{EUF}$-conflict : $((v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \neg(v_6 = v_7) \wedge (v_2 = v_5)) \rightarrow \bot$

$\mathcal{LA}(\mathbb{Q})$-deduction : $((v_2 = v_3 - v_4) \wedge (v_5 = 0) \wedge (v_3 = v_4)) \rightarrow (v_2 = v_5)$

$\mathcal{EUF}$-deduction : $((v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_0 = v_1)) \rightarrow (v_3 = v_4)$

$\mathcal{LA}(\mathbb{Q})$-deduction : $((v_0 \geq v_1) \wedge (v_0 \leq v_1)) \rightarrow (v_0 = v_1)$

$\Longrightarrow$

$\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$-conflict : $((v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \neg(v_6 = v_7) \wedge (v_2 = v_3 - v_4)$

                        $\wedge(v_5 = 0) \wedge (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_0 \geq v_1)) \rightarrow \bot.$

4.2.2 No learning from e_{ij}-reasoning

Second, the conflict(ing) clauses above cannot provide any information about the theory-combination steps performed by the $\mathcal{T}_1 \cup \mathcal{T}_2$-*solver*. Thus, in future branches, if run on very similar set of of literals, the combined $\mathcal{T}_1 \cup \mathcal{T}_2$-solver may have to repeat part or all the same e_{ij}-deduction steps.

Example 7 Consider the scenario at the end of Example 5 (Fig. 4). Thus, DPLL backtracks on the conflict found in Example 6 and assigns false to $RESET_5$,[11] resulting in the new assignment $\mu' \stackrel{\text{def}}{=} \mu_{\mathcal{EUF}} \cup \mu'_{\mathcal{LA}(\mathbb{Q})}$ s.t.

$$\begin{aligned} \mu_{\mathcal{EUF}} = & \{(v_3 = h(v_0)), (v_4 = h(v_1)), (v_6 = f(v_2)), (v_7 = f(v_5)), \neg(v_6 = v_7)\} \\ \mu'_{\mathcal{LA}(\mathbb{Q})} = & \{(v_0 \leq v_1), (v_0 \geq v_1), (v_2 = v_3 - v_4)(v_5 = v_8)\}, \end{aligned} \quad (6)$$

which is found $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$-satisfiable with a similar process (see Fig. 4), in which the e_{ij}-deductions of $(v_0 = v_1)$ and $(v_3 = v_4)$ have to be performed again. Notice that the conflict clause found in Example 6 provides no help to avoid repeating the two e_{ij}-deduction steps in the right branch in Fig. 4.

4.2.3 Forcing internal case-splits in non-convex theories

Third, in case of non-convex theories, the combined $\mathcal{T}_1 \cup \mathcal{T}_2$-*solver* must handle internally the case-splits caused by the fact that each $\mathcal{T}_i$-*solver* may receive from the other disjunctions of interface equalities.

Example 8 Consider the conjunction of literals $\mu \stackrel{\text{def}}{=} \mu_{\mathcal{EUF}} \wedge \mu_{\mathcal{LA}(\mathbb{Z})}$ (2) of Example 1:

$$\begin{aligned} \mu_{\mathcal{EUF}} : & \ \neg(f(v_1) = f(v_2)) \wedge \neg(f(v_2) = f(v_4)) \wedge (f(v_3) = v_5) \wedge (f(v_1) = v_6) \wedge \\ \mu_{\mathcal{LA}(\mathbb{Z})} : & \ (v_1 \geq 0) \wedge (v_1 \leq 1) \wedge (v_5 = v_4 - 1) \wedge (v_3 = 0) \wedge (v_4 = 1) \\ & \wedge (v_2 \geq v_6) \wedge (v_2 \leq v_6 + 1). \end{aligned} \quad (7)$$

Here all the variables $(v_1, \ldots, v_6)$ are interface ones. μ contains only unit clauses, so after the first run of unit-propagations, DPLL generates the assignment μ which is simply the set of literals in μ. One possible run of the NO procedure is depicted in Fig. 5.[12]

First, the sub-assignment $\mu_{\mathcal{EUF}}$ is given to the $\mathcal{EUF}$-solver, which reports its consistency and deduces no interface equality. Then, the sub-assignment $\mu_{\mathcal{LA}(\mathbb{Z})}$ is given to the $\mathcal{LA}(\mathbb{Z})$-solver, which reports its consistency and deduces first $(v_3 = v_5)$ and then the disjunction $(v_1 = v_3) \vee (v_1 = v_4)$, which are both passed to the $\mathcal{EUF}$-solver. Whilst the first produces no effect, the second forces a case-splitting so that the two equalities $(v_1 = v_3)$ and $(v_1 = v_4)$ must be analyzed separately by the $\mathcal{EUF}$-solver. The first branch, corresponding to selecting $(v_1 = v_3)$, is opened: then the set $\mu_{\mathcal{EUF}} \cup \{(v_1 = v_3)\}$ is $\mathcal{EUF}$-consistent, and the equality $(v_5 = v_6)$ is deduced. After that, the assignment $\mu_{\mathcal{LA}(\mathbb{Z})} \cup \{(v_5 = v_6)\}$ is passed to the $\mathcal{LA}(\mathbb{Z})$-solver, that reports its consistency and deduces another disjunction, $(v_2 = v_3) \vee (v_2 = v_4)$. At this point, another case-splitting is needed in the $\mathcal{EUF}$-solver, resulting in the two branches $\mu_{\mathcal{EUF}} \cup \{(v_1 = v_3), (v_2 = v_3)\}$ and $\mu_{\mathcal{EUF}} \cup \{(v_1 = v_3), (v_2 = v_4)\}$. Both of them are found inconsistent, so the whole branch previously opened by the selection of $(v_1 = v_3)$ is found inconsistent.

---

[11] We assume that DPLL adopts the last UIP strategy using the $\mathcal{T}$-conflict clause described in Example 6. Other strategies may lead to propagate also $\neg(v_5 = 0)$ in the right branch, which would not affect the result.

[12] Notice that there may be different runs depending on the order in which the e_{ij}-deductions are performed.
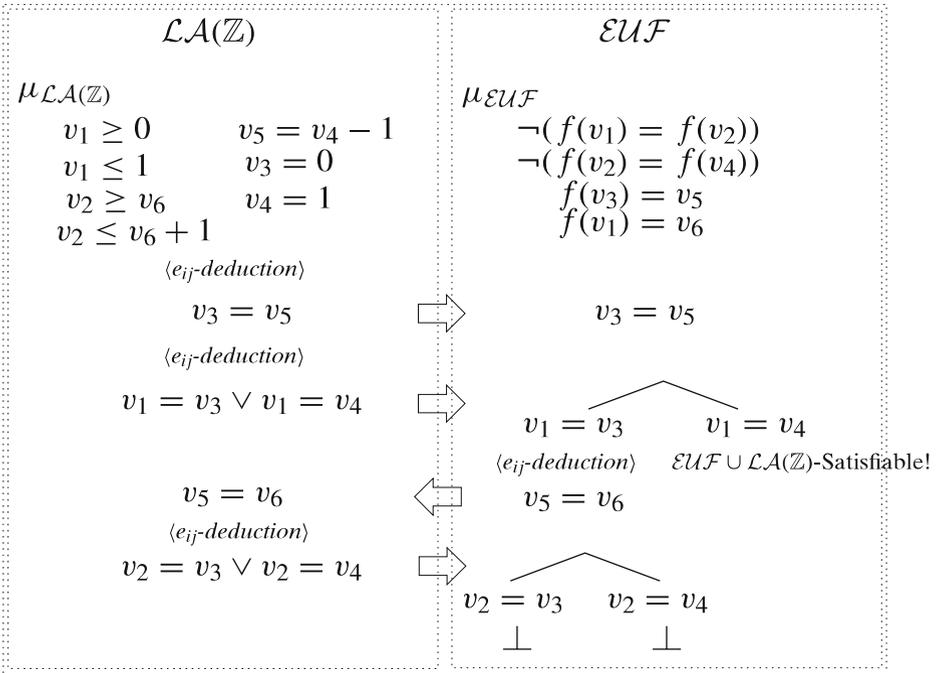
Fig. 5 The NO search tree for the formula of Example 8

At this point, the other case of the branch (i.e., the equality $(v_1 = v_4)$) is selected, and since the assignment $\mu_{\mathcal{EUF}} \cup \{(v_1 = v_4)\}$ is $\mathcal{EUF}$-consistent and no new interface equality is deduced, the NO method reports the $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Z})$-satisfiability of μ.

4.2.4 Requiring e_{ij}-deduction complete $\mathcal{T}_i$-solvers

Finally, as highlighted from the two previous examples, the ability of the $\mathcal{T}_i$-solvers to carry out e_{ij}-deductions (see Section 3.2) is crucial: each solver must be e_{ij}-deduction complete, that is, it must be able to derive the (disjunctions of) interface equalities e_{ij} which are entailed by its current facts $\mu_{\mathcal{T}_i}$. As highlighted in Section 3.2, for some theories this operation can be very expensive.

Remark 1 For the sake of simplicity and better readability, we have provided only a high-level description of N.O., which does not consider the optimizations and implementation techniques which have been proposed in the literature (see [5, 17, 21, 30, 37, 38] for more details). To the best of our knowledge, however, all optimizations and implementation techniques for N.O. introduced before DTC are orthogonal to the issues raised in Sections 4.2.1–4.2.4.

5 SMT for combined theories via the DTC procedure

Delayed Theory Combination (DTC) is a more recent general-purpose procedure for tackling the problem of theory combination directly in the context of lazy *SMT*

[9, 10]. DTC works by performing Boolean reasoning on interface equalities, possibly combined with $\mathcal{T}$-propagation, with the help of the embedded DPLL solver. As with N.O. procedure, DTC is based on the N.O. logical framework of Section 2.2, and thus considers signature-disjoint stably-infinite theories with their respective $\mathcal{T}_i$-solvers, and pure input formulas (although most of the considerations on releasing purity and stably-infiniteness in Section 2.2 hold for DTC as well). Importantly, no assumption is made about the e_{ij}-deduction capabilities of the $\mathcal{T}_i$-solvers (Section 3.2): for each $\mathcal{T}_i$-solver, every intermediate situation from complete e_{ij}-deduction to no e_{ij}-deduction capabilities is admitted.

5.1 The DTC procedure

A basic architectural schema of DTC is described in Fig. 6. In DTC, each of the two $\mathcal{T}_i$-solvers interacts directly and only with the Boolean enumerator (DPLL), so that there is no direct exchange of information between the $\mathcal{T}_i$-solvers. The Boolean enumerator is instructed to assign truth values not only to the atoms in $Atoms(\varphi)$, but also to the interface equalities e_{ij}'s. Consequently, each assignment μ^p enumerated by DPLL is partitioned into three components $\mu^p_{\mathcal{T}_1}$, $\mu^p_{\mathcal{T}_2}$ and μ^p_e, s.t. each $\mu_{\mathcal{T}_i}$ is the set of i-pure literals and μ_e is the set of interface (dis)equalities in μ, so that each $\mu_{\mathcal{T}_i} \cup \mu_e$ is passed to the respective $\mathcal{T}_i$-solver.

An implementation of DTC [9, 10] is based on the online schema of Fig. 1 in Section 3.1, exploiting early pruning, $\mathcal{T}$-propagation, $\mathcal{T}$-backjumping and $\mathcal{T}$-learning. Each of the two $\mathcal{T}_i$-solvers interacts with the DPLL engine by exchanging literals via the assignment μ in a stack-based manner. The $\mathcal{T}$-DPLL algorithm of Fig. 1 in Section 3.1 is modified to the following extents:

1. $\mathcal{T}$-DPLL is instructed to assign truth values not only to the atoms in φ, but also to the interface equalities not occurring in φ. $\mathcal{B}2\mathcal{T}$ and $\mathcal{T}2\mathcal{B}$ are modified accordingly.

Fig. 6 A basic architectural schema of $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$ via the DTC procedure
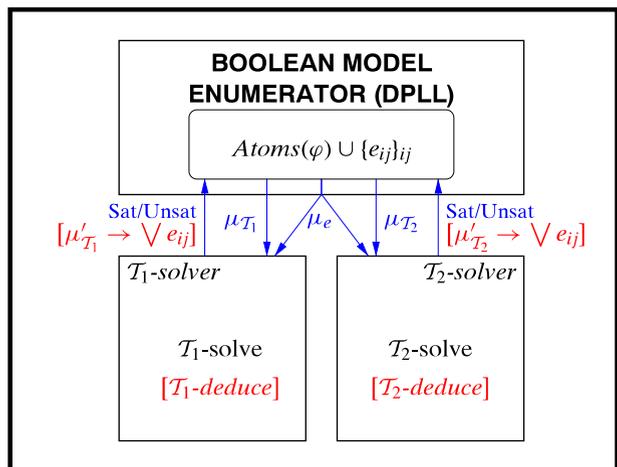
2. $\mathcal{T}$-decide_next_branch is modified to select also interface equalities e_{ij}'s not occurring in the formula yet,[13] but only after the current assignment propositionally satisfies φ.

3. $\mathcal{T}$-deduce is modified to work as follows: instead of feeding the whole μ to a (combined) $\mathcal{T}$-solver, for each $\mathcal{T}_i$, $\mu_{\mathcal{T}_i} \cup \mu_e$, is fed to the respective $\mathcal{T}_i$-*solver*. If both return sat, then $\mathcal{T}$-deduce returns sat, otherwise it returns Conflict.

4. $\mathcal{T}$-analyze_conflict and $\mathcal{T}$-backtrack are modified so that to use the conflict set returned by one $\mathcal{T}_i$-*solver* for $\mathcal{T}$-backjumping and $\mathcal{T}$-learning. Importantly, such conflict sets may contain interface (dis)equalities.

5. Early-pruning and $\mathcal{T}$-propagation are performed. If one $\mathcal{T}_i$-solver performs the e_{ij}-deduction $\mu^* \models_{\mathcal{T}_i} \bigvee_{j=1}^{k} e_j$ s.t. $\mu^* \subseteq \mu_{\mathcal{T}_i} \cup \mu_e$ and each e_j is an interface equality, then the deduction clause $\mathcal{T}2\mathcal{B}(\mu^* \to \bigvee_{j=1}^{k} e_j)$ is learned.

6. **[If and only if both $\mathcal{T}_i$-*solvers* are e_{ij}-deduction complete.]** If an assignment μ which propositionally satisfies φ is found $\mathcal{T}_i$-satisfiable for both $\mathcal{T}_i$'s, and neither $\mathcal{T}_i$-*solver* performs any e_{ij}-deduction from μ, then $\mathcal{T}$-DPLL stops returning sat.[14]

In order to achieve efficiency, other heuristics and strategies have been further suggested in [9, 10], and more recently in [15, 19].

In short, in DTC the embedded DPLL engine not only enumerates truth assignments for the atoms of the input formula, but it also assigns truth values for the interface equalities that the $\mathcal{T}$-*solvers* are not capable of inferring, and handles the case-split induced by the entailment of disjunctions of interface equalities in non-convex theories. The rationale is to exploit the full power of a modern DPLL engine and to delegate to it part of the heavy reasoning effort on interface equalities previously due to the $\mathcal{T}_i$-*solvers*.

5.2 Discussion

DTC has been conceived in such a way to fully exploit the power of DPLL within a lazy SMT framework. We analyze the behaviour of DTC with the help of the examples we used in Section 4.2 for N.O., considering both the case in which the $\mathcal{T}_i$-*solvers* are e_{ij}-deduction complete and the case in which the $\mathcal{T}_i$-*solvers* have no e_{ij}-deduction capability, with both convex and non-convex theories.

In all the following examples we assume that DTC adopts the "N.O.-mimicking strategy" of Fig. 7, which will be discussed in Section 6. Moreover, in order to simplify the explanation and to introduce some concepts which will be elaborated in Section 6, in both Examples 9 and 11 ($\mathcal{T}_i$-*solvers* with no e_{ij}-deduction capability) we will assume that both $\mathcal{T}_i$-*solver* always return $\neg e_{ij}$-minimal conflict sets.

Notationally, $\mu'_{\mathcal{T}_i}$, $\mu''_{\mathcal{T}_i}$, $\mu'''_{\mathcal{T}_i}$ denote generic subsets of $\mu_{\mathcal{T}_i}$ and "C_{ij}" denotes either the $\mathcal{T}$-deduction clause causing the $\mathcal{T}$-propagation of $(v_i = v_j)$ or the conflicting clause causing the backjump to $(v_i = v_j)$. For better readability, we represent directly

---

[13]Notice that an interface equality occurs in the formula after a clause containing it is learned, see point 4.

[14]This is identical to the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability termination condition of N.O. procedure.

Strategy 1 ("NO-mimicking")

1. *All the conflict clauses derived by theory conflicts are learned, either temporarily or permanently.*
2. *Each conflict clause in 1. is a mixed Boolean+theory conflict clause which is built from the theory conflict set by means of the last-UIP strategy described in §3.1.*
3. *The literal selection heuristic and the $\mathcal{T}_i$-solverscalls are such that:*

   (i) *e_{ij}'s not occurring in the formula (included learned clauses) are selected only when the current branch already propositionally satisfies the input formula;*
   (ii) *Early pruning (EP) is applied before every selection of a new e_{ij},*
   (iii) *the new e_{ij}'s selected are always assigned a negative value,*
   (iv) *each $\mathcal{T}_i$-solveris invoked only if at least one literal (which has not been deduced singularly by $\mathcal{T}_i$-solveritself) has been added to its input since the last call.* [15]

4. *At every early-pruning call on a branch (namely μ) which is found both $\mathcal{T}_1$- and $\mathcal{T}_2$-consistent, if one $\mathcal{T}_i$-solver performs the e_{ij}-deduction $\mu^* \models_{\mathcal{T}_i} \bigvee_{j=1}^{k} e_j$, s.t. $\mu^* \subseteq \mu_{\mathcal{T}_i}$, each e_j being an unassigned interface equality on variables in μ, then:*

   (i) *the clause $\mathcal{T}2\mathcal{B}(\mu^* \to \bigvee_{j=1}^{k} e_j)$ is learned immediately;*
   (ii) *if $k = 1$, then e_k is added to the current assignment and unit-propagated immediately;*
   (iii) *if $k > 1$, then $\neg e_1, ..., \neg e_k$ are put on the top of the literal selection list, so that to be the next $\neg e_{ij}$'s selected by the literal selection heuristic.*

5. **[If and only if both $\mathcal{T}_i$-*solvers*are e_{ij}-deduction complete]**
   If an assignment μ which propositionally satisfies φ is found $\mathcal{T}_i$-satisfiable for both $\mathcal{T}_i$'s, and neither $\mathcal{T}_i$-solver performs any e_{ij}-deduction from μ, then DTC stops returning sat*.*

Fig. 7 A "NO-mimicking" strategy for DTC

the assignment to $\mathcal{T}$-atoms rather than to their Boolean abstraction (e.g., we say "assign $\neg(v_5 = 0)$" instead of "assign $\neg B_i$" s.t. $B_i \overset{\text{def}}{=} \mathcal{T}2\mathcal{B}((v_5 = 0))$).

5.2.1 Delaying theory combination

First, thanks to the fact that $\mathcal{T}$-decide_next_branch can select new interface equalities after μ propositionally satisfies φ (point 2. above), the Boolean search tree is divided into two parts: the top part, performed on the atoms currently occurring in the formula, in which a (partial) truth assignment μ propositionally satisfying φ is searched, and the bottom part, performed on the e_{ij}'s which do not yet occur in the formula, in which the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of μ is checked by building a candidate arrangement μ_e. Thus, in every branch the reasoning on e_{ij}'s is not performed until (and unless) it is strictly necessary. (From which the name "Delayed Theory

---

[15]This avoids invoking a $\mathcal{T}_i$-*solver* twice in sequence on the same input. The restriction "which ... by $\mathcal{T}_i$-*solver* itself" means that, if $\mathcal{T}_i$-*solver* (μ) returns "Sat" and deduces e_{ij}, then $\mathcal{T}_i$-*solver* is not invoked on $\mu \cup \{e_{ij}\}$.

Combination".) E.g., if in one branch μ is such that one $\mu_{\mathcal{T}_i}$ component is $\mathcal{T}_i$-unsatisfiable, no Boolean reasoning on e_{ij}'s is performed.

To this extent, it is important to exploit the issue of partial assignments [36]: when the current partial assignment μ propositionally satisfies the input formula φ, the remaining atoms occurring in φ can be ignored and only the new e_{ij}'s are then selected. Importantly, if μ is a partial assignment, then it is sufficient that μ_e assigns only the e_{ij}'s which have an actual interface role in μ. (E.g., if μ is partial and v is an interface variable in φ but it occurs in no 1-pure literal in μ, then v has no "interface role" for μ, so that every interface equality containing v can be ignored by μ_e.)

Example 9 Consider again the $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$-formula φ (4) of Example 5:

$$
\begin{array}{ll}
\mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\
\mathcal{LA}(\mathbb{Q}): & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \qquad (8) \\
Both: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).
\end{array}
$$

and consider the assignment μ (5) obtained after $\mathcal{T}$-DPLL assigns $RESET_5$ and unit-propagates $(v_5 = 0)$. Let μ be partitioned into $\mu_{\mathcal{LA}(\mathbb{Q})}$ and $\mu_{\mathcal{EUF}}$ as in Fig. 8. μ propositionally satisfies φ ($\mu \models_p \varphi$), and μ is a partial assignment because it does not assign $(v_5 = v_8)$. By a call to the $\mathcal{T}_i$-solvers, both $\mu_{\mathcal{LA}(\mathbb{Q})}$ and $\mu_{\mathcal{EUF}}$ are found consistent in the respective theories. Thus, in order to check the $\mathcal{T}_1 \cup \mathcal{T}_2$-consistency of μ, $\mathcal{T}$-DPLL generates and explores a Boolean search sub-trees on the e_{ij}s according to the Strategy of Fig. 7.

First $\mathcal{T}$-DPLL starts selecting (the negated value of) the new e_{ij}'s, each time invoking incrementally the $\mathcal{T}_i$-solvers (EP), until it selects $\neg(v_0 = v_1)$, which causes
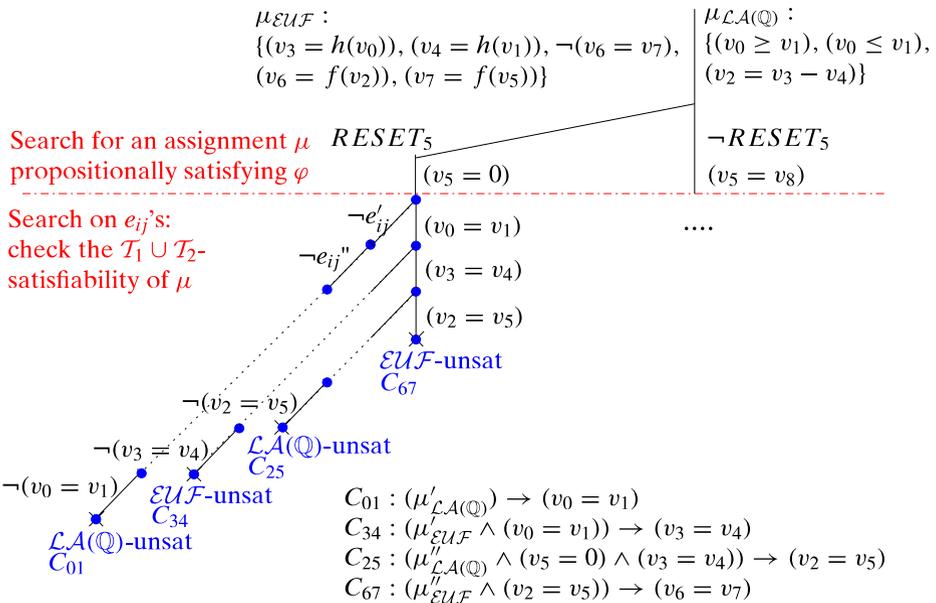
Fig. 8 DTC execution of the first branch of Example 9, with no e_{ij}-deduction. Here we assume that all conflict sets returned by the $\mathcal{T}_i$-solvers are $\neg e_{ij}$-minimal

a $\mathcal{LA}(\mathbb{Q})$ conflict. As $\mathcal{LA}(\mathbb{Q})$ is convex and $\mathcal{LA}(\mathbb{Q})$-Solver is $\neg e_{ij}$-minimal, it returns a conflict set in the form $\mu'_{\mathcal{LA}(\mathbb{Q})} \cup \{\neg(v_0 = v_1)\}$ s.t. $\{(v_0 \geq v_1), (v_0 \leq v_1)\} \subseteq \mu'_{\mathcal{LA}(\mathbb{Q})} \subseteq \mu_{\mathcal{LA}(\mathbb{Q})}$. Thus DTC learns the corresponding clause C_{01} and backjumps up to μ (or even higher), hence unit propagating $(v_0 = v_1)$.

What happens next depends on whether the learned clause C_{01} contains the redundant $\mathcal{LA}(\mathbb{Q})$ atom $(v_5 = 0)$ or not. Here we consider the "worst" case, when such atom occurs in C_{01}. This means that DTC backjumps after the unit-propagation of $(v_5 = 0)$.[16] Then $(v_0 = v_1)$ is unit-propagated and new unassigned $\neg e_{ij}$'s are selected again, until $\neg(v_3 = v_4)$ generates another conflict represented by clause C_{34}, which causes backjumping and unit-propagating $(v_3 = v_4)$. The same is repeated for $(v_2 = v_5)$. Then $\mu \cup \{(v_0 = v_1), (v_3 = v_4), (v_2 = v_5)\}$ is found $\mathcal{EUF}$-inconsistent s.t. the conflict is represented by the clause C_{67}, and the whole procedure backtracks, causing the unit-propagation of $\neg RESET_5$ and $(v_5 = v_8)$.

Then the search proceeds from here, with the benefit that $\mathcal{T}$-DPLL can reuse the clauses C_{01}-C_{67} to avoid repeating research performed in the previous branch, as explained below.

5.2.2 Learning from e_{ij} reasoning

Second, thanks to points 1., 2., 4. and 5., the interface equalities e_{ij}'s are included in the conflict(ing) and deduction clauses derived by $\mathcal{T}$-conflicts and $\mathcal{T}$-deduction. Therefore, instead of one long e_{ij}-free $\mathcal{T}_1 \cup \mathcal{T}_2$-conflict clause, it is possible to learn several (much shorter) conflicts and deduction clauses corresponding to the conflicts and deductions returned by the $\mathcal{T}_i$-solvers. Moreover, the reasoning steps on e_{ij}'s which are performed in order to decide the $\mathcal{T}_1 \cup \mathcal{T}_2$-consistency of one branch μ (both Boolean search on e_{ij}'s and e_{ij}-deduction steps) are saved in the form of clauses and thus they can be reused to check the $\mathcal{T}_1 \cup \mathcal{T}_2$-consistency of all subsequent branches. This allows from pruning search and prevents redoing the same search/deduction steps from scratch.

Example 10 Consider again the $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$ formula φ of Examples 5 and 9. Figure 9 illustrates a DTC execution when both $\mathcal{T}_i$-solvers are e_{ij}-deduction complete (that is, under the same hypotheses as NO). As before, we assume $\mathcal{T}$-DPLL adopts Strategy 1 of Fig. 7.

On the left branch (when $RESET_5$ is selected), after $(v_5 = 0)$ is unit-propagated, the $\mathcal{LA}(\mathbb{Q})$-solver deduces $(v_0 = v_1)$, and thus the deduction clause C_{01} is learned and $(v_0 = v_1)$ is unit-propagated. Consequently, the $\mathcal{EUF}$-solver can deduce $(v_3 = v_4)$, causing the learning of C_{34} and the unit-propagation of $(v_3 = v_4)$, which in turn causes the $\mathcal{LA}(\mathbb{Q})$-deduction of $(v_2 = v_5)$, the learning of C_{25} and the unit-propagation of $(v_2 = v_5)$.

At this point, $\mu''_{\mathcal{EUF}} \cup \{(v_2 = v_5)\}$ is found $\mathcal{EUF}$-inconsistent, so that the $\mathcal{EUF}$-solver returns (the negation of) the clause C_{67}, which is resolved backward with the clauses C_{25}, C_{34}, C_{01}, and $(RESET_5 \rightarrow (v_5 = 0))$ forcing DTC to backjump

---

[16]More precisely, by Step 2. of Strategy 1, DTC eliminates $(v_5 = 0)$ from the conflict clause C_{01} by resolving the latter with the clause $RESET_5 \rightarrow (v_5 = 0)$ in φ, thus substituting $(v_5 = 0)$ with $RESET_5$ into the conflict clause used to drive $\mathcal{T}$-backjumping. A similar process happens in the next steps.

$\mu_{\mathcal{EUF}}:$
$\{(v_3 = h(v_0)), (v_4 = h(v_1)), \neg(v_6 = v_7),$
$(v_6 = f(v_2)), (v_7 = f(v_5))\}$

$\mu_{\mathcal{LA}(\mathbb{Q})}:$
$\{(v_0 \geq v_1), (v_0 \leq v_1),$
$(v_2 = v_3 - v_4)\}$

$RESET_5$ $\neg RESET_5$

Search for an assignment μ propositionally satisfying φ

$\mathcal{LA}(\mathbb{Q})$-deduce $(v_0 = v_1)$ $(v_5 = 0)$ $(v_5 = v_8)$
   C_{01}

$\mathcal{EUF}$-deduce $(v_3 = v_4)$ $(v_0 = v_1)$ $(v_0 = v_1)$
   C_{34}

$\mathcal{LA}(\mathbb{Q})$-deduce $(v_2 = v_5)$ $(v_3 = v_4)$ $(v_3 = v_4)$
   C_{25} $(v_2 = v_5)$

Satisfiable

Search on e_{ij}'s: check the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of μ

$\mathcal{EUF}$-unsat
C_{67}

$C_{01} : (\mu'_{\mathcal{LA}(\mathbb{Q})}) \to (v_0 = v_1)$
$C_{34} : (\mu'_{\mathcal{EUF}} \wedge (v_0 = v_1)) \to (v_3 = v_4)$
$C_{25} : (\mu''_{\mathcal{LA}(\mathbb{Q})} \wedge (v_5 = 0) \wedge (v_3 = v_4)) \to (v_2 = v_5)$
$C_{67} : (\mu''_{\mathcal{EUF}} \wedge (v_2 = v_5)) \to (v_6 = v_7)$
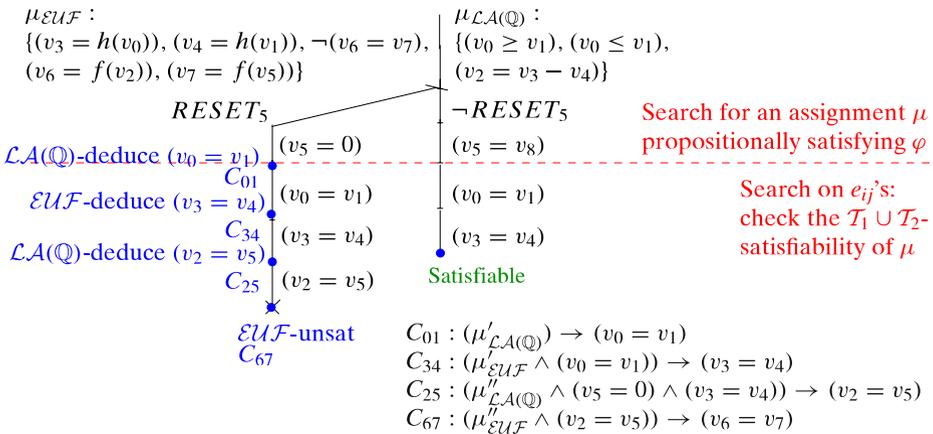
Fig. 9 DTC execution of Example 10, with e_{ij}-deduction-complete $\mathcal{T}_i$-solvers

up to the last branching point and to unit-propagate $\neg RESET_5$. Hence $(v_5 = v_8)$ is unit-propagated on the clause $\neg RESET_5 \to (v_5 = v_8)$, which produces another assignment propositionally satisfying φ.

Then, $(v_0 = v_1)$ and hence $(v_3 = v_4)$ are unit-propagated on C_{01} and C_{34} respectively, with no need to call the $\mathcal{T}$-solvers.[17] At this point, since neither $\mathcal{T}_i$-solver can deduce any new e_{ij}, by step 6. DTC concludes that φ is $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$-satisfiable.

We notice that in the right branch of the DTC search tree, all values are assigned directly by unit-propagation. Thus, DTC "remembers" in form of clauses the e_{ij}-deductions performed in the first branch and reuses them in the subsequent branch so that to avoid redoing them from scratch.

A similar situation happens in the right branch of Example 9.

5.2.3 Allowing lack of e_{ij}-deduction capability

Third, DTC allows for using $\mathcal{T}_i$-solvers with partial or no e_{ij}-deduction capability, because part of or all the e_{ij}-deductions can be substituted by extra Boolean search on the e_{ij}'s performed by the DPLL engine. Vice versa, if the $\mathcal{T}_i$-solvers do have some or full e_{ij}-deduction capability, DTC exploits this fact by means of $\mathcal{T}$-propagation. Thus, by adopting $\mathcal{T}$-solvers with different e_{ij}-deduction power, one can trade part or all the (possibly very expensive) e_{ij}-deduction effort for extra Boolean search.

Example 11 (*See also Example 9*) Consider again the conjunction of literals $\mu \stackrel{\text{def}}{=} \mu_{\mathcal{EUF}} \wedge \mu_{\mathcal{LA}(\mathbb{Z})}$ of Examples 1 and 8. We assume here that both the $\mathcal{EUF}$- and $\mathcal{LA}(\mathbb{Z})$-solver's have no e_{ij}-deduction capabilities, and that they always return conflict sets which do not contain redundant negated interface equalities. One possible session of DTC is depicted in Fig. 10.

---

[17]Here we assume for simplicity that $\mu'_{\mathcal{LA}(\mathbb{Q})}$ in C_{01} does not contain the redundant literal $(v_5 = 0)$. If this is not the case, one more $\mathcal{T}$-propagation of $(v_0 = v_1)$ is needed.

$$\mu_{\mathcal{EUF}}: \quad \mu_{\mathcal{LA}(\mathbb{Z})}:$$

$$\begin{array}{ll}
\neg(f(v_1) = f(v_2)) & v_1 \geq 0 \qquad v_5 = v_4 - 1 \\
\neg(f(v_2) = f(v_4)) & v_1 \leq 1 \qquad v_3 = 0 \\
f(v_3) = v_5 & v_2 \geq v_6 \qquad v_4 = 1 \\
f(v_1) = v_6 & v_2 \leq v_6 + 1
\end{array}$$

Search for an assignment μ propositionally satisfying φ

$v_3 = v_5$

Search on e_{ij}'s: check the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of μ

$\neg(v_3 = v_5)$

$\mathcal{LA}(\mathbb{Z})$-unsat, C_{35}

$v_1 = v_4$

$\neg(v_1 = v_4)$

Satisfiable

$v_1 = v_3$

$\neg(v_1 = v_3)$

$v_5 = v_6$

$\mathcal{LA}(\mathbb{Z})$-unsat, C_{13}

$v_2 = v_4$

$\neg(v_5 = v_6)$

$\mathcal{EUF}$-unsat, C_{14}

$\mathcal{EUF}$-unsat, C_{56}

$\neg(v_2 = v_4)$

$v_2 = v_3$

$\neg(v_2 = v_3)$ $\quad \mathcal{EUF}$-unsat, C_{24}

$\mathcal{LA}(\mathbb{Z})$-unsat, C_{23}

$$C_{35} : \mu'_{\mathcal{LA}(\mathbb{Z})} \rightarrow (v_3 = v_5)$$
$$C_{13} : (\mu'_{\mathcal{LA}(\mathbb{Z})}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$$
$$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$$
$$C_{23} : (\mu''_{\mathcal{LA}(\mathbb{Z})} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4))$$
$$C_{24} : (\mu''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_3)) \rightarrow \bot$$
$$C_{14} : (\mu'''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_4)) \rightarrow \bot$$

Fig. 10 DTC execution of Example 11 on $\mathcal{LA}(\mathbb{Z}) \cup \mathcal{EUF}$, with no e_{ij}-deduction. Here we assume that all conflict sets returned by the $\mathcal{T}_i$-solvers are $\neg e_{ij}$-minimal

Initially, both $\mu_{\mathcal{LA}(\mathbb{Z})}$ and $\mu_{\mathcal{EUF}}$ are found consistent in each of the theories by the respective $\mathcal{T}_i$-solvers. Then $\mathcal{T}$-DPLL starts selecting new $\neg e_{ij}$'s, and proceeds without causing conflicts, until it selects $\neg(v_3 = v_5)$, which causes a $\mathcal{LA}(\mathbb{Z})$ conflict on the conflicting clause C_{35}, and forces $\mathcal{T}$-DPLL to backjump and to unit-propagate $(v_3 = v_5)$.

Then $\mathcal{T}$-DPLL selects new $\neg e_{ij}$'s until it selects $\neg(v_1 = v_4)$ and $\neg(v_1 = v_3)$, which cause a $\mathcal{LA}(\mathbb{Z})$ conflict. The branch is in the form $\mu \cup \bigcup_j \neg e_j$, so that, the $\neg e_{ij}$-minimal conflict set η_{13} returned is in the form $\mu'_{\mathcal{LA}(\mathbb{Z})} \cup \{\neg(v_1 = v_3), \neg(v_1 = v_4)\}$ s.t. $\mu'_{\mathcal{LA}(\mathbb{Z})}$ contain no other negated interface equality. Thus $\mathcal{T}$-DPLL uses the corresponding clause C_{13} (see Fig. 10) to backjump up to the highest point which allows for unit-propagating $(v_1 = v_3)$ on C_{13}, and performs such unit propagation. Then $\mathcal{T}$-DPLL proceeds selecting new $\neg e_{ij}$'s without causing conflicts, until it selects $\neg(v_5 = v_6)$, which causes a $\mathcal{EUF}$ conflict represented by the clause C_{56}. As $\mathcal{EUF}$ is convex, $\neg(v_5 = v_6)$ is the only $\neg e_{ij}$ occurring in the conflict set, so that $\mathcal{T}$-DPLL backtracks over the last chain of $\neg e_{ij}$'s and unit-propagates $(v_5 = v_6)$.

Again, $\mathcal{T}$-DPLL selects a chain of new $\neg e_{ij}$'s until it selects $\neg(v_2 = v_4)$ and $\neg(v_2 = v_3)$, which cause a $\mathcal{LA}(\mathbb{Z})$ conflict represented by clause C_{23}. As before, it backjumps to the highest point where it can unit-propagate $(v_2 = v_3)$. Performing the latter unit propagation causes a $\mathcal{EUF}$ conflict (clause C_{24}). By applying Step 2. of Strategy 1, resolving on literal $(v_2 = v_3)$ the conflicting clause C_{24} with the clause C_{23} (which caused the unit-propagation of $(v_2 = v_3)$), $\mathcal{T}$-DPLL backjumps over all the remaining $\neg e_{ij}$'s of the current chain and unit-propagating $(v_2 = v_4)$.

The latter causes a new $\mathcal{EUF}$ conflict represented by the clause C_{14}. Again, by Step 2. of Strategy 1, C_{14} is resolved with the clauses C'_{24}, C_{56}, C_{13} (which caused the

propagation of $(v_2 = v_4)$, $(v_5 = v_6)$, $(v_1 = v_3)$ respectively), backjumping up to μ and unit-propagating $(v_1 = v_4)$.

Finally, $\mathcal{T}$-DPLL starts and proceeds selecting $\neg e_{ij}$'s (possibly unit-propagating some value due to the clauses learned) without generating conflicts, so that to conclude that the formula is $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable.

5.2.4 Handling case-splits in non-convex theories

Finally, in case of non-convex theories, in DTC the case-splits caused by the deduction of disjunctions of e_{ij}'s by the $\mathcal{T}_i$-solvers are handled directly by the DPLL engine. (Notice that, unlike with the "splitting on demand" approach of [3], here we refer to the case-splits which are necessary to *handle* the deductions performed by the other $\mathcal{T}_i$-solver, rather to those which may be necessary to *perform* such deductions.)

Example 12 (*See also Example 11*) Consider the $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Z})$ formula φ and assignment μ of Example 8. Figure 11 illustrates a DTC execution when both $\mathcal{T}_i$-solvers are e_{ij}-deduction complete. As before, we assume $\mathcal{T}$-DPLL adopts Strategy 1 of Fig. 7.
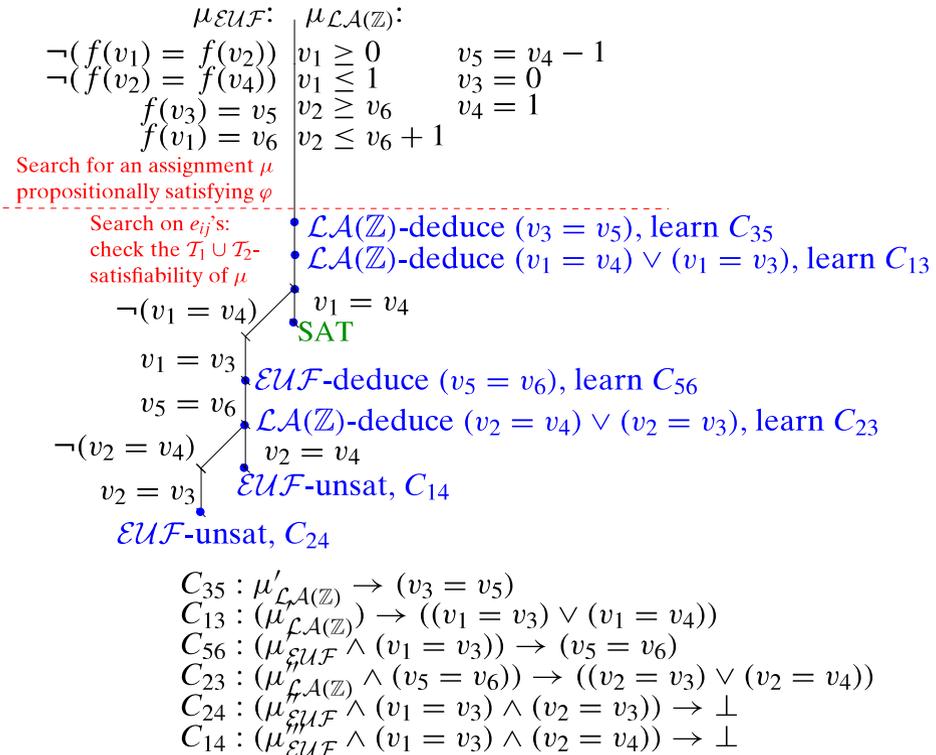
Fig. 11 DTC execution of Example 12 on $\mathcal{LA}(\mathbb{Z}) \cup \mathcal{EUF}$, with e_{ij}-deduction-complete $\mathcal{T}_i$-solvers

The first invocation of the $\mathcal{LA}(\mathbb{Z})$ solver results in deducing $(v_3 = v_5)$ and the disjunction $(v_1 = v_4) \vee (v_1 = v_3)$ and in learning of the corresponding clauses C_{35} and C_{13}. By Step 4.(iii) of Strategy 1, then, $(v_1 = v_4)$ and $(v_1 = v_3)$ are put on the top of the literal selection list. As a consequence, DTC selects $\neg(v_1 = v_4)$, and thanks to C_{13} it immediately unit-propagates $(v_1 = v_3)$. At this point the $\mathcal{EUF}$ solver can deduce $(v_5 = v_6)$, so that the clause C_{56} is learned and the deduced equality is unit-propagated immediately. When $\mu_{\mathcal{LA}(\mathbb{Z})} \cup \{(v_5 = v_6)\}$ is passed to the $\mathcal{LA}(\mathbb{Z})$ solver, this deduces the disjunction $(v_2 = v_4) \vee (v_2 = v_3)$, learning C_{23}. Selecting $\neg(v_2 = v_4)$ results in the unit-propagation of $(v_2 = v_3)$, which in turn causes a $\mathcal{EUF}$ conflict. After the $\mathcal{EUF}$-solver returns (the negation of) C_{24}, DTC backjumps up to a point where $(v_2 = v_4)$ can be unit-propagated. This results again in an $\mathcal{EUF}$-conflict, so that the $\mathcal{EUF}$-solver returns (the negation of) C_{14}, which causes another backjumping up to where $(v_1 = v_4)$ can be unit-propagated. Then, after another invocation to the theory solvers, DTC stops, declaring φ to be $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Z})$-satisfiable.

6 Controlling the enlargement of the Boolean search space

The benefits of DTC wrt. NO highlighted in Section 5 come at the potential cost of an enlargement of the Boolean search space explored by $\mathcal{T}$-DPLL. To this extent, in this section, we analyze the enlargement of the Boolean search space in DTC wrt. NO procedure, and prove the following facts.

1. Under the same working hypotheses of NO procedure (stably-infinite theories, incremental, backtrackable and e_{ij}-deduction-complete $\mathcal{T}_i$-*solvers*), there is a "NO-mimicking" strategy for DTC s.t. no extra Boolean search on the e_{ij}'s is performed wrt. N.O procedure: in case of convex theories, no extra Boolean search on e_{ij}'s is performed; in case on non-convex theories, the only Boolean search on e_{ij}'s performed is that caused by the case-splits induced by the disjunctions of e_{ij}'s (which NO procedure must perform internally to the combined $\mathcal{T}_1 \cup \mathcal{T}_2$-*solver*).
2. If some $\mathcal{T}_i$-*solver* is not e_{ij}-deduction complete, then the "NO-mimicking" strategy for DTC mimics the e_{ij}-deductions performed by NO procedure via $\mathcal{T}$-backjumping, and the cost in terms of Boolean search can be controlled in terms of the "quality" of the $\mathcal{T}$-conflict sets η returned by the $\mathcal{T}_i$-*solvers*: the more redundant $\neg e_{ij}$'s are removed from η, the more branches are pruned; if the η's contain no redundant $\neg e_{ij}$, then the Boolean search reduces to only one branch for every e_{ij}-deduction mimicked.

Result 1 states that, under the same working hypotheses, the DTC procedure is "at least as good as the NO procedure" in terms of Boolean search space. Result 2 states that, if the $\mathcal{T}_i$-*solver* have partial or no e_{ij}-deduction capabilities, then the amount of extra Boolean search required can be reduced down to a negligible amount by reducing the presence of redundant negated disequalities in the $\mathcal{T}$-conflict sets returned by the $\mathcal{T}$-*solvers*.

The NO-mimicking strategy for DTC is described in Fig. 7. For convenience, we prove results 2 and 1 in reverse order, in Sections 6.2 and 6.1 respectively.

6.1 DTC with non e_{ij}-deduction-complete $\mathcal{T}_i$-solvers vs. NO

In this section, we assume that both the $\mathcal{T}_i$-solvers employed by DTC are $\neg e_{ij}$-minimal and have limited or no e_{ij}-deduction capabilities. Under these assumptions, we have the following result.

Theorem 2 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two stably-infinite (possibly non-convex) theories. Let both $\mathcal{T}_i$-solvers be $\neg e_{ij}$-minimal, and possibly have some e_{ij}-deduction capabilities; let φ be a pure $\mathcal{T}_1 \cup \mathcal{T}_2$ formula and let μ be a total assignment propositionally satisfying φ. Let DTC with Strategy 1 prove the $\mathcal{T}_1 \cup \mathcal{T}_2$-consistency (resp. $\mathcal{T}_1 \cup \mathcal{T}_2$-inconsistency) of μ, returning a conflict set η in the case of inconsistency. Let dtc_br and dtc_ded be the number of Boolean branches and of e_{ij}-deductions performed in the DTC proof. Then we have:*

$$dtc\_br + dtc\_ded \le no\_br + no\_ded, \tag{9}$$

no_ded and no_br being respectively the number of deductions and of branches performed by a corresponding NO proof of the $\mathcal{T}_1 \cup \mathcal{T}_2$-consistency (resp. $\mathcal{T}_1 \cup \mathcal{T}_2$-inconsistency) of μ.

Proof We consider a generical branch μ s.t. μ propositionally satisfies the input formula φ. We reason by induction on the structure of the DTC Boolean search tree required to prove the $\mathcal{T}_1 \cup \mathcal{T}_2$-unsatisfiability of μ.

Base We have two basic cases (Fig. 12, top).

1. Let μ be $\mathcal{T}_i$-unsatisfiable for some $\mathcal{T}_i$. The $\mathcal{T}_i$-solver detects this fact returning an $\neg e_{ij}$-minimal conflict set η. Thus $dtc\_br = 1$ and $dtc\_ded = 0$. Similarly, in every N.O. refutation $\mathcal{T}_i$-solver detects the $\mathcal{T}_i$-unsatisfiability of μ. Thus $no\_ded = 0$ and $no\_br = 1$, so that (9) holds.
2. Let μ be $\mathcal{T}_i$-satisfiable for both $\mathcal{T}_i$'s, and no (disjunction of) e_{ij}'s can be deduced from μ. DTC selects a chain of new negated e_{ij}'s, invoking an early-pruning check before each new selection which cause no e_{ij}-deduction, until no new e_{ij}'s are available, from which it concludes that μ is $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable. Here $dtc\_br = 1$ and $dtc\_ded = 0$. In every corresponding N.O. proof, both $\mathcal{T}_i$-solvers return "Sat" without performing e_{ij}-deductions (i.e., $no\_br = 1$, $no\_ded = 0$). Therefore (9) holds.

Step If none of the previous cases holds, then DTC selects a chain of new negated e_{ij}'s, invoking an early-pruning check before each new selection, until either (Fig. 12, bottom):

(a) one early-pruning call to one $\mathcal{T}_i$-solver returns Unsat. In this case, let B denote the current branch. The $\mathcal{T}_i$-solver returns also a $\neg e_{ij}$-minimal conflict set η, corresponding to the conflicting clause

$$\eta^* \to \bigvee_{j=1}^{k} e_j, \tag{10}$$
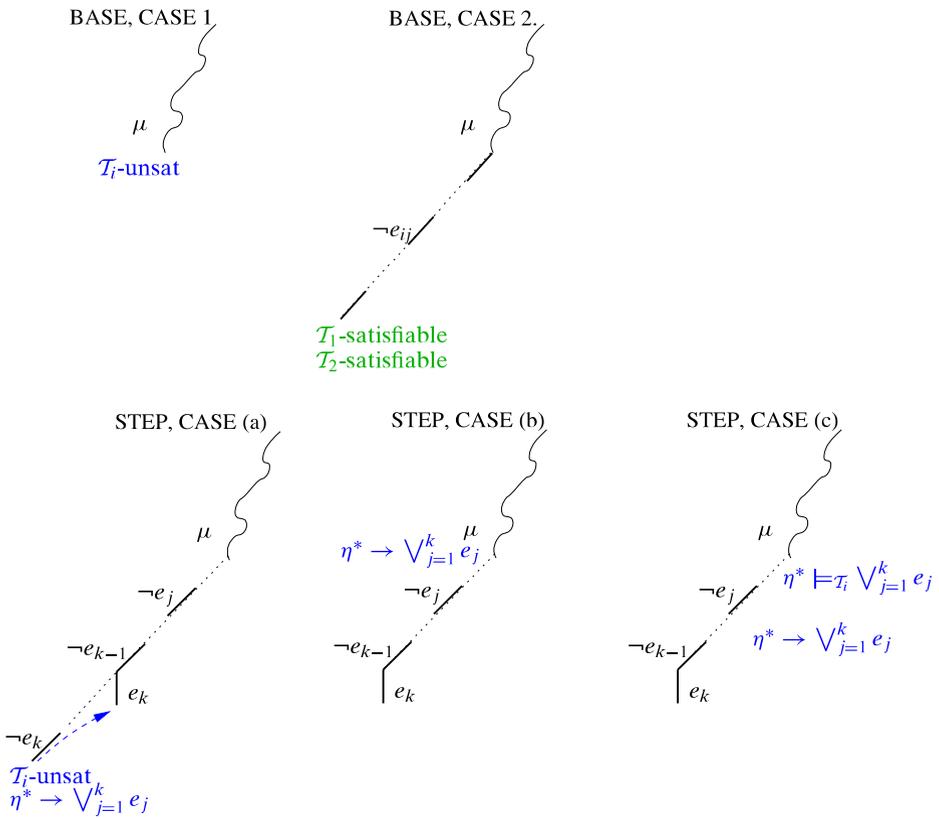
Fig. 12 Graphical representations of base cases 1. and 2. (*1st row*) and Step cases (a), (b), (c) (*2nd row*)

where $\neg e_1, ..., \neg e_k$ are all the $\neg e_{ij}$'s occurring in η, and $\eta^* \stackrel{\text{def}}{=} \eta \setminus \{\neg e_1, ..., \neg e_k\}$. (In the corresponding N.O. refutation, this corresponds to the e_{ij}-deduction $\mu \models_{\mathcal{T}_i} \bigvee_{j=1}^k e_j$.) DTC learns the conflicting clause (10) and backjumps, popping up the literals from the branch (if $k > 1$) up to $\neg e_{k-1}$ or (if $k = 1$) up to the highest point in μ where e_k can be unit-propagated on (10), and hence unit propagating e_k. (In the N.O. refutation, this corresponds to selecting the branch $\mu \cup \{e_k\}$.)

(b) one positive[18] e_{ij}, namely e_k, is unit-propagated due to some previously-learned conflict clause, which we can write w.l.o.g. in the form (10), s.t. all $\neg e_1, ..., \neg e_{j-1}$

---

[18] The case where one *negative* e_{ij} is unit-propagated due to some previously-learned conflict clause C does not affect the overall discussion, because it will be eliminated from every conflict clause by means of of Step 2. in Strategy 1. No literal other than (negated) e_{ij}'s can be unit propagated, because μ assigns a truth value to all the atomic expressions in φ.
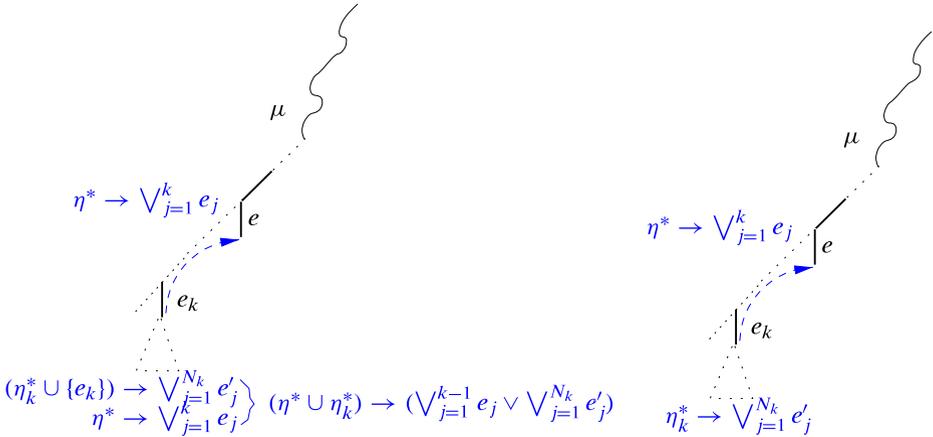
Fig. 13 Graphical representation of the recursive behaviour. Case 1 (*left*) and 2 (*right*)

and all the literals in η^* are in the current branch, and η^* contains no negated e_{ij}'s. (Notice that, in the N.O. refutation, this might require a novel e_{ij}-deduction $\mu \models_{\mathcal{T}_i} \bigvee_{j=1}^{k} e_j$.[19])

(c) One $\mathcal{T}_i$-*solver* performs a e_{ij}-deduction, namely $\eta^* \models_{\mathcal{T}_i} \bigvee_{j=1}^{k} e_j$, s.t. η^* is part of the current branch and the e_j's are not. By Step 4. of Strategy 1, the clause (10) is learned immediately, DTC selects in order $\neg e_1, ..., \neg e_{k-1}$, and hence it unit-propagates e_k on clause (10). (In the corresponding N.O. refutation, this corresponds to the e_{ij}-deduction $\eta^* \models_{\mathcal{T}_i} \bigvee_{j=1}^{k} e_j$ and to the selection of the branch e_k.)

In each of the three cases above, let $B_{e_k} \stackrel{\text{def}}{=} \mu_k \cup \{e_k\}$ be the current branch, s.t. $\mu_k \supseteq \mu$ and $\neg e_j \in \mu_k$ for every $j < k$. Now DTC checks recursively the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of B_{e_k}. We have that $B_{e_k} \models_p \varphi$ and $B_{e_k} \supset \mu$, so that the subtree below B_{e_k} is a strict subtree of that below μ, so that we can apply the inductive hypothesis to B_{e_k}.

If B_{e_k} is recursively found $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable, then DTC concludes that μ is $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable.

Otherwise, B_{e_k} is recursively found $\mathcal{T}_1 \cup \mathcal{T}_2$-unsatisfiable. By inductive hypothesis, the DTC sub-proof requires dtc_ded_k e_{ij}-deductions and dtc_br_k branches, whilst the N.O. subproof requires no_ded_k e_{ij}-deductions and no_br_k branches, s.t. $dtc\_br_k + dtc\_ded_k \leq no\_br_k + no\_ded_k$. Let η_k be the conflict set returned and let N_k be the

---

[19]E.g., the $\mathcal{EUF}$ deduction of $(v_3 = v_4)$ in the right branch of Fig. 4 corresponds to a simple unit-propagation on clause C_{34} in Fig. 8.

number of negated equalities in η_k. We distinguish two subcases (Fig. 13, left and right):

1. $e_k \in \eta_k$. Thus $\eta_k = \eta_k^* \cup \{\neg e_j'\}_{j=1}^{N_k} \cup \{e_k\}$, s.t. η_k^* does not contain $\neg e_{ij}$'s. By Step 2. of Strategy 1, DTC eliminates e_k from the conflicting clause $\neg \eta_k$ by resolving it with (10), obtaining the new conflict clause:

$$(\eta^* \cup \eta_k^*) \rightarrow \left(\bigvee_{j=1}^{k-1} e_j \vee \bigvee_{j=1}^{N_k} e_j' \right). \tag{11}$$

Let $\neg e$ be the most recently assigned $\neg e_{ij}$ in $\{\neg e_j\}_{j=1}^{k-1} \cup \{\neg e_j'\}_{j=1}^{N_k}$. Then DTC backjumps up to the highest point in B_{e_k} where e can be unit-propagated on (11), and hence unit-propagates e. (Notice that (11) dominates (10) in driving the backjumping mechanism because all the $\neg e_j'$'s occur higher in B than $\neg e_k$.)

2. $e_k \notin \eta_k$. Thus $\eta_k = \eta_k^* \cup \{\neg e_j'\}_{j=1}^{N_k}$, s.t. η_k^* does not contain $\neg e_{ij}$'s, corresponding to the clause:

$$\eta_k^* \rightarrow \bigvee_{j=1}^{N_k} e_j'. \tag{12}$$

Let $\neg e$ be the most recently assigned $\neg e_{ij}$ in $\{\neg e_j'\}_{j=1}^{N_k}$. Then DTC backjumps up to the highest point in B_{e_k} where e can be unit-propagated on (12), and hence unit-propagates e. (Notice that also (12) dominates (10) in driving the backjumping mechanism because all the $\neg e_j'$'s occur higher in B than $\neg e_k$.)

Then, DTC proceeds, each time checking recursively the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability on one open branch $B_{e_j} \stackrel{\text{def}}{=} \mu_j \cup \{e_j\}$, each e_j corresponding to either one of the original negated e_{ij}'s $\neg e_1, ..., \neg e_k$ in B, or to one of the negated e_{ij}'s occurring in the conflict sets reported by the recursive sub-proofs. This is done until either a subbranch is recursively found to be $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable, or the current dominating conflict clause forces DTC backjumping up to a point within μ, so that μ can be declared $\mathcal{T}_1 \cup \mathcal{T}_2$-unsatisfiable, and the negation of the dominating clause is the conflict set returned.

Let N be the number of sub-proofs performed. By inductive hypothesis, the j-th DTC sub-proof requires dtc_br_j branches and dtc_ded_j e_{ij}-deductions, whilst the corresponding N.O. sub-proof requires no_ded_j e_{ij}-deductions and no_br_j branches, s.t. $dtc\_br_j + dtc\_ded \leq no\_br_j + no\_ded_j$. In the cases (a)–(c) above we have respectively:

Case (a): $dtc\_br = 1 + \sum_{j=1}^{N} dtc\_br_j$, $dtc\_ded = \sum_{j=1}^{N} dtc\_ded_j$, $no\_br = \sum_{j=1}^{N} no\_br_j$, and $no\_ded = 1 + \sum_{j=1}^{N} no\_ded_j$.

Case (b): $no\_br = \sum_{j=1}^{N} no\_br_j$, $dtc\_ded = \sum_{j=1}^{N} dtc\_ded_j$, $dtc\_br = \sum_{j=1}^{N} dtc\_br_j$, and either $no\_ded = \sum_{j=1}^{N} no\_ded_j$ or $no\_ded = 1 + \sum_{j=1}^{N} no\_ded_j$.

Case (c): $dtc\_br = \sum_{j=1}^{N} dtc\_br_j$, $dtc\_ded = 1 + \sum_{j=1}^{N} dtc\_ded_j$, $no\_br = \sum_{j=1}^{N} no\_br_j$, and $no\_ded = 1 + \sum_{j=1}^{N} no\_ded_j$.

In all cases, (9) holds. □

Theorem 2 states that, if the $\mathcal{T}_i$-solvers are both $\neg e_{ij}$-minimal, then there is a strategy for DTC which emulates some NO proof (even though the $\mathcal{T}_i$-solvers have

limited or no e_{ij}-deduction capabilities!) at the cost of (at most) one extra Boolean branch for every e_{ij}-deduction performed by NO. Therefore the (possibly very expensive) e_{ij}-deduction steps of the NO schema can be avoided at the cost of one extra Boolean branch each.

More generally, we notice that one key idea in the proof of Theorem 2 is that, when the DPLL engine detects an inconsistency and generates a conflict set η, it backjumps up to the second-most-recently-assigned $\neg e_{ij}$ in η, if any. (See, e.g., the case of C_{23} in Fig. 10.) Therefore, in a more general case than that of Theorem 2 (no $\neg e_{ij}$-minimality), *the more redundant $\neg e_{ij}$'s the $\mathcal{T}_i$-solvers are able to remove from the conflict set returned, the more Boolean branches are skipped by backjumping*.

Example 13 (*convex case*) Compare the behaviour of NO and of DTC in the first branch of Figs. 4 and 8 of Examples 5 and 9 respectively. We notice that in DTC the whole process mimics the NO deduction process of the first branch in Example 5, requiring a number of extra Boolean branches equal to the number of deductions performed by the corresponding NO process ($dtc\_br = 4$, $dtc\_ded = 0$, $no\_br = 1$ and $no\_ded = 3$. Notice that the three leftmost diagonal branches in Fig. 8 obtain the same effect as the e_{ij}-deduction steps in Fig. 4 (and in Fig. 9).

Example 14 (*non-convex case*) Compare the behaviour of NO and of DTC in Figs. 5 and 10 of Examples 8 and 11 respectively. Again, we notice that in DTC the whole process mimics the NO deduction process in Example 5, in the sense that the backjumping steps on the clauses C_{35}, C_{13}, C_{56}, and C_{23} mimic the effects of performing the corresponding e_{ij}-deductions in Fig. 5. Overall, we notice that in Fig. 10 DTC explores only seven branches, four for mimicking the corresponding e_{ij}-deductions and three for mimicking the three case-split branches in Fig. 5 ($dtc\_br = 7$, $dtc\_ded = 0$, $no\_br = 3$ and $no\_ded = 4$).

6.2 DTC with e_{ij}-deduction-complete $\mathcal{T}_i$-solvers vs. NO

In this section, we assume that both the $\mathcal{T}_i$-solvers employed by DTC are e_{ij}-deduction complete. Under these assumptions, we have the following result.

Theorem 3 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two stably-infinite (possibly non-convex) theories and let both $\mathcal{T}_i$-solvers be e_{ij}-deduction complete; let φ be a pure $\mathcal{T}_1 \cup \mathcal{T}_2$ formula and let μ be an assignment propositionally satisfying φ. Let DTC with Strategy 1 prove the $\mathcal{T}_1 \cup \mathcal{T}_2$-consistency (resp. $\mathcal{T}_1 \cup \mathcal{T}_2$-inconsistency) of μ, returning a conflict set η in the case of inconsistency. Let dtc_br be the number of Boolean branches required in the DTC proof. Then we have:*

$$dtc\_br \leq no\_br \qquad (13)$$

no_br being the number of branches performed by a corresponding NO proof of the $\mathcal{T}_1 \cup \mathcal{T}_2$-consistency (resp. $\mathcal{T}_1 \cup \mathcal{T}_2$-inconsistency) of μ.

Proof As before, we consider a generical branch μ s.t. μ propositionally satisfies the input formula φ, and we reason by induction on the structure of the DTC Boolean search tree required to prove the $\mathcal{T}_1 \cup \mathcal{T}_2$-unsatisfiability of μ.

Base Let μ be $\mathcal{T}_i$-unsatisfiable for some $\mathcal{T}_i$'s. In this case, the proof is as for Theorem 2.

Let μ be $\mathcal{T}_i$-satisfiable for both $\mathcal{T}_i$'s, and neither $\mathcal{T}_i$-solver can perform any e_{ij}-deduction from μ. Then by step 5. of Strategy 1, DTC can conclude that μ if $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable. The same would do every NO tool. Thus $dtc\_br = no\_br = 1$, so that (13) holds.

Step If some previously-learned clauses forces some unit-propagation, the proof is as for Theorem 2.

If none of the two base conditions hold, and no unit propagation can be performed on μ, then μ is $\mathcal{T}_i$-satisfiable for both $\mathcal{T}_i$'s and one $\mathcal{T}_i$-solver performs a deduction, namely $\mu \models_{\mathcal{T}_i} \bigvee_{j=1}^{k} e_j$. By Step 4. of Strategy 1, the clause (10) is learned immediately, and hence DTC selects in order $\neg e_1, ..., \neg e_{k-1}$, and hence it unit-propagates e_k on clause (10). (In the corresponding NO refutation, this corresponds to the e_{ij}-deduction $\mu \models_{\mathcal{T}_i} \bigvee_{j=1}^{k} e_j$ and to the selection of the branch e_k.)

Henceforth, the proof is as for Theorem 2, except for the fact that, by induction hypothesis, $dtc\_br_k \leq no\_br_k$, and $dtc\_br := \sum_{j=1}^{N} dtc\_br_j$, $no\_br := \sum_{j=1}^{N} no\_br_j$ for every j. Thus (13) holds. □

Theorem 3 states that, under the same hypotheses of e_{ij}-deduction as NO, DTC mimics NO with no extra cost in terms of Boolean search.

Example 15 (*convex case*) Compare the behaviour of NO and of DTC in the first branch of Figs. 4 and 9 of Examples 5 and 10 respectively. We notice that in DTC the whole process mimics the NO deduction process of the first branch in Example 5, requiring no extra Boolean branches and the same number of deductions performed by the corresponding NO process ($dtc\_br = 1$, $dtc\_ded = 3$, $no\_br = 1$ and $no\_ded = 3$). (The main difference relies on the fact that, unlike with NO, the deduced e_{ij}'s are not exchanged directly by the $\mathcal{T}_i$-*solvers*; rather, they are added to the current assignment μ and unit-propagated.)

In the right branch, instead, in DTC all values are assigned directly by unit-propagation, saving two deductions wrt. NO ($dtc\_br = 1$, $dtc\_ded = 0$, $no\_br = 1$ and $no\_ded = 2$).

Example 16 (*non-convex case*) Compare the behaviour of NO and of DTC in Figs. 5 and 11 of Examples 8 and 12 respectively. Again, we notice that in DTC the whole process mimics the NO deduction process in Example 8, requiring a number of Boolean branches corresponding to the branches performed internally to the $\mathcal{T}_1 \cup \mathcal{T}_2$-*solver* in NO ($dtc\_br = 3$, $no\_br = 3$) and and the same number of deductions performed by the corresponding NO process ($dtc\_ded = 4$, $no\_ded = 4$).

7 Conclusions

In this paper we have presented a detailed comparative analysis of the NO and the DTC procedures for $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$. Our analysis has highlighted some important advantages of DTC in exploiting the power of modern lazy DPLL-based SAT solvers: first, DTC naturally allows for learning clauses containing interface

equalities, which can be used in subsequent branches to prune search and avoid redoing the same search/deductions from scratch; second, it does not require exhaustive deduction capabilities to the $\mathcal{T}_i$-solvers, although it can fully exploit them by means of $\mathcal{T}$-propagation; third, it nicely encompasses the case of non-convex theories, by delegating to the embedded DPLL solver the task of handling the case-splits causes by the non-convexity of the theories.

As far as the possible increase of Boolean search space, we have shown that, by exploiting the full power of advanced SMT techniques like Early Pruning, $\mathcal{T}$-propagation, $\mathcal{T}$-backjumping and $\mathcal{T}$-learning, there is a strategy which allows DTC to mimic the behavior of NO, so that:

(i)   under the same hypotheses of e_{ij}-deduction capabilities of the $\mathcal{T}_i$-solvers required by NO, DTC requires no extra Boolean search (Theorem 3);

(ii) using $\mathcal{T}_i$-solvers with limited or no e_{ij}-deduction capabilities, the extra Boolean search required can be reduced down to a negligible amount by controlling the quality of the $\mathcal{T}$-conflict sets returned by the $\mathcal{T}$-solvers (Theorem 2).

We remark that the NO-mimicking strategy has been conceived only for the sake of proving Theorems 2 and 3 (e.g., last UIP is used instead of the more efficient 1st UIP in order to mimic the naive case-splits performed by NO, see Section 4.2.3), and that the strategies implemented by actual tools introduce further improvements in terms of efficiency. E.g., in MATHSAT [8, 13] 1st UIP is used, which typically allows for a stronger pruning of the Boolean search space [45], Step 3.(ii) is substituted with a weakened version of EP [8], which reduces the effort of repeated calls to the $\mathcal{T}_i$-solvers, and more effective literal-selection strategies are preferred to Step 3.(i) and (iii).

As far as the $\neg e_{ij}$-minimality hypothesis is concerned, we notice that, at least for theories like $\mathcal{EUF}$ and $\mathcal{LA}(\mathbb{Q})$, there are known decision procedures that fulfill this requirement (see [31] and [8] respectively.) For other theories, the problem of $\neg e_{ij}$-minimization opens a novel research branch.[20] However, we remark that DTC works also when the $\mathcal{T}_i$-solvers are not $\neg e_{ij}$-minimal, at the cost of (at most) one extra branch to explore for each redundant $\neg e_{ij}$ returned in a conflict set.

On the whole, the results presented in this paper show that DTC allows for trading Boolean search for e_{ij}-deduction. Thus everyone can choose and implement the most suitable $\mathcal{T}_i$-solvers without being forced by the e_{ij}-deduction-completeness straitjacket: for theories for which efficient e_{ij}-deduction complete procedures are available (e.g., $\mathcal{EUF}$ [31]), DTC allows for exploiting the full power of e_{ij}-deduction; for harder theories (e.g., $\mathcal{LA}(\mathbb{Z})$), the research task changes from that of finding e_{ij}-deduction complete $\mathcal{T}$-solvers to that of finding $\neg e_{ij}$-minimal or nearly-$\neg e_{ij}$-minimal ones.

---

[20]Bottom line, one can always make μ $\neg e_{ij}$-minimal by dropping the remaining $\neg e_{ij}$'s one by one, each time checking $\mu \setminus \{\neg e_{ij}\}$. Notice that, in general, with $\neg e_{ij}$-minimization the search for the candidate $\neg e_{ij}$'s to drop is restricted to only those occurring in μ, whilst with e_{ij}-deduction the search for the candidate e_{ij}'s to deduce extends to all the unassigned e_{ij}'s.

References

1. Ball, T., Cook, B., Lahiri, S.K., Zhang, L.: Zapato: automatic theorem proving for predicate abstraction refinement. In: Proc. CAV'04. LNCS, vol. 3114. Springer, New York (2004)
2. Barrett, C., Berezin, S.: CVC Lite: a new implementation of the cooperating validity checker. In: Proceedings of the 16th International Conference on Computer Aided Verification (CAV '04). LNCS, vol. 3114. Springer, New York (2004)
3. Barrett, C., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Splitting on demand in SAT modulo theories. In: Proc. LPAR'06. LNAI, vol. 4246. Springer, New York (2006)
4. Barrett, C., Tinelli, C.: Cvc3. In: Proc. CAV'07. LNCS, vol. 4590. Springer, New York (2007)
5. Barrett, C.W., Dill, D.L., Stump, A.: A generalization of Shostak's method for combining decision procedures. In: Frontiers of Combining Systems (FROCOS). Lecture Notes in Artificial Intelligence. Springer, Santa Margherita Ligure (2002)
6. Bonacina, M.P., Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Decidability and undecidability results for Nelson-Oppen and rewrite-based decision procedures. In: Proc. of IJCAR'06. LNAI, no. 4130 (2006)
7. Bozzano, M., Bruttomesso, R., Cimatti, A., Franzen, A., Hanna, Z., Khasidashvili, Z., Palti, A., Sebastiani, R.: Encoding RTL constructs for MathSAT: a preliminary report. In: Proc. PDPAR'05. ENTCS, vol. 144. Elsevier, Amsterdam (2006)
8. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., Rossum, P., Schulz, S., Sebastiani, R.: An incremental and layered procedure for the satisfiability of linear arithmetic logic. In: Proc. TACAS'05. LNCS, vol. 3440. Springer, New York (2005)
9. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., van Rossum, P., Ranise, S., Sebastiani, R.: Efficient satisfiability modulo theories via delayed theory combination. In: Proc. CAV 2005. LNCS, vol. 3576. Springer, New York (2005)
10. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., van Rossum, P., Ranise, S., Sebastiani, R.: Efficient theory combination via boolean search. Inf. Comput. **204**(10), 1493–1525 (2006)
11. Brinkmann, R., Drechsler, R.: RTL-datapath verification using integer linear programming. In: Proc. ASP-DAC 2002, pp. 741–746. IEEE, Piscataway (2002)
12. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: Delayed theory combination vs. Nelson-Oppen for satisfiability modulo theories: a comparative analysis. In: Proc. LPAR'06. LNAI, vol. 4246. Springer, New York (2006)
13. Bruttomesso, R., Cimatti, A., Franzen, A., Griggio, A., Sebastiani, R.: The MathSAT 4 SMT solver. In: CAV. LNCS, vol. 5123. Springer, New York (2008)
14. Cotton, S., Maler, O.: Fast and flexible difference logic propagation for DPLL(T). In: Proc. SAT'06. LNCS, vol. 4121. Springer, New York (2006)
15. de Moura, L., Bjørner, N.: Model-based theory combination. In: Proc. of the 5th Workshop on Satisfiability Modulo Theories SMT'07. http://www.lsi.upc.edu/~oliveras/smt07/ (2007)
16. de Moura, L., Owre, S., Ruess, H., Rushby, J., Shankar, N.: The ICS decision procedures for embedded deduction. In: Proc. IJCAR'04. LNCS, vol. 3097, pp. 218–222. Springer, New York (2004)
17. Detlefs, D., Nelson, G., Saxe, J.: Simplify: a theorem prover for program checking. J. ACM **52**(3), 365–473 (2005)
18. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Proc. CAV'06. LNCS, vol. 4144. Springer, New York (2006)
19. Dutertre, B., de Moura, L.: System description: Yices 1.0. In: Proc. on 2nd SMT competition, SMT-COMP'06. yices.csl.sri.com/yices-smtcomp06.pdf (2006)
20. Enderton, H.: A Mathematical Introduction to Logic. Academic, London (1972)
21. Filliâtre, J.-C., Owre, S., Rueß, H., Shankar, N.: ICS: Integrated Canonizer and Solver. In: Proc. CAV'2001 (2001)
22. Flanagan, C., Joshi, R., Ou, X., Saxe, J.B.: Theorem proving using lazy proof explication. In: Proc. CAV 2003. LNCS. Springer, New York (2003)
23. Fontaine, P., Ranise, S., Zarba, C.G.: Combining lists with non-stably infinite theories. In: Proc. LPAR'04. LNCS, vol. 3452. Springer, New York (2004)
24. Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: DPLL(T): fast decision procedures. In: Proc. CAV'04. LNCS, vol. 3114, pp. 175–188. Springer, New York (2004)
25. Ghilardi, S.: Model theoretic methods in combined constraint satisfiability. J. Autom. Reason. **33**(3), 221–249 (2004)
26. Ghilardi, S., Nicolini, E., Zucchelli, D.: A comprehensive framework for combined decision procedures. In: Proc. FroCos'05. LNCS, vol. 3717. Springer, New York (2005)

27. Krstic, S., Goel, A.: Architecting solvers for SAT modulo theories: Nelson-Oppen with DPLL. In: Proc. Frontiers of Combining Systems, 6th International Symposium, FroCoS 2007. LNAI, vol. 4720. Springer, New York (2007)
28. Krstić, S., Goel, A., Grundy, J., Tinelli, C.: Combined satisfiability modulo parametric theories. In: TACAS'07. LNCS, vol. 4424. Springer, New York (2007)
29. Lahiri, S.K., Musuvathi, M.: An efficient decision procedure for UTVPI constraints. In: Proc. of 5th International Workshop on Frontiers of Combining Systems (FroCos '05). LNCS, vol. 3717. Springer, New York (2005)
30. Nelson, C.G., Oppen, D.C.: Simplification by cooperating decision procedures. TOPLAS **1**(2), 245–257 (1979)
31. Nieuwenhuis, R., Oliveras, A.: Congruence closure with integer offsets. In: Proc. 10th LPAR. LNAI, no. 2850, pp. 77–89. Springer, New York (2003)
32. Nieuwenhuis, R., Oliveras, A.: DPLL(T) with exhaustive theory propagation and its application to difference logic. In: Proc. CAV'05. LNCS, vol. 3576. Springer, New York (2005)
33. Oppen, D.C.: Complexity, convexity and combinations of theories. Theor. Comp. Sci. **12**, 291–302 (1980)
34. Ranise, S., Ringeissen, C., Zarba, C.G.: Combining data structures with nonstably infinite theories using many-sorted logic. In: Proc FroCos'05. LNCS, vol. 3717. Springer, New York (2005)
35. Rueß, H., Shankar, N.: Deconstructing Shostak. In: Proc. LICS '01. IEEE Computer Society, Piscataway (2001)
36. Sebastiani, R.: Lazy satisfiability modulo theories. Journal on Satisfiability, Boolean Modeling and Computation, JSAT. **3**, 141–224 (2007)
37. Shankar, N., Rueß, H.: Combining Shostak theories. Invited paper for Floc'02/RTA'02 (2002)
38. Shostak, R.: A pratical decision procedure for arithmetic with function symbols. J. ACM **26**(2), 51–360 (1979)
39. Shostak, R.: Deciding combinations of theories. J. ACM **31**, 1–12 (1984)
40. Tinelli, C., Harandi, M.T.: A new correctness proof of the Nelson–Oppen combination procedure. In: Proc. Frontiers of Combining Systems, FroCoS'06. Applied Logic. Kluwer, Dordrecht (1996)
41. Tinelli, C., Ringeissen, C.: Unions of non-disjoint theories and combinations of satisfiability procedures. Theor. Comp. Sci. **290**(1), 291–353 (2003)
42. Tinelli, C., Zarba, C.: Combining nonstably infinite theories. J. Autom. Reason. **34**(3), 209–238 (2005)
43. Zarba, C.G.: A tableau calculus for combining non-disjoint theories. In: Proc. Tableaux'02. Lecture Notes in Computer Science, vol. 2381, pp. 315–329. Springer, New York (2002)
44. Zarba, C.G.: Combining sets with integers. In: FroCos'02. Lecture Notes in Computer Science, vol. 2309, pp. 103–116. Springer, New York (2002)
45. Zhang, L., Madigan, C.F., Moskewicz, M.H., Malik, S.: Efficient conflict driven learning in a boolean satisfiability solver. In: Proc. ICCAD '01. IEEE, Piscataway (2001)
46. Zhang, L., Malik, S.: The quest for efficient boolean satisfiability solvers. In: Proc. CAV'02. LNCS, no. 2404, pp. 17–36. Springer, New York (2002)