Delayed Theory Combination vs. Nelson-Oppen for Satisfiability Modulo Theories: a Comparative Analysis*

Roberto Bruttomesso[1], Alessandro Cimatti[1], Anders Franzén[1,2], Alberto Griggio[2], and Roberto Sebastiani[2]

[1] ITC-IRST, Povo, Trento, Italy. {bruttomesso,cimatti,franzen}@itc.it
[2] DIT, Università di Trento, Italy. {griggio,rseba}@dit.unitn.it

Abstract. Many approaches for Satisfiability Modulo Theory ($SMT(\mathcal{T})$) 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*). When $\mathcal{T}$ is the combination $\mathcal{T}_1 \cup \mathcal{T}_2$ of two simpler theories, the approach is typically handled by means of Nelson-Oppen's (NO) theory combination schema in which two specific $\mathcal{T}$*-solvers* deduce and exchange (disjunctions of) interface equalities.

In recent papers we have proposed a new approach to $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$, called *Delayed Theory Combination* (DTC). Here part or all the (possibly very expensive) task of deducing interface equalities is played by the SAT solver itself, at the potential cost of an enlargement of the boolean search space. In principle this enlargement could be up to exponential in the number of interface equalities generated.

In this paper we show that this estimate was too pessimistic. We present a comparative analysis of DTC vs. NO for $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$, which shows that, using state-of-the-art SAT-solving techniques, the amount of boolean branches performed by DTC can be upper bounded by the number of deductions and boolean branches performed by NO on the same problem. We prove the result for different deduction capabilities of the $\mathcal{T}$*-solvers* and for both convex and non-convex theories.

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 $\mathcal{T}$. Theories of interest for many applications are, e.g., the theory of difference logic $\mathcal{DL}$, the theory $\mathcal{EUF}$ of equality and uninterpreted functions, the quantifier-free fragment of Linear Arithmetic over the rationals $\mathcal{LA}(\mathbb{Q})$ and that over the integers $\mathcal{LA}(\mathbb{Z})$. Particularly relevant is the case of $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$, where the background theory $\mathcal{T}$ is the combination of two (or more) simpler theories $\mathcal{T}_1$ and $\mathcal{T}_2$. [3]

* This work has been partly supported by ISAAC, an European sponsored project, contract no. AST3-CT-2003-501848, by ORCHID, a project sponsored by Provincia Autonoma di Trento, and by a grant from Intel Corporation.

[3] For better readability, and as it is common practice in papers dealing with combination of theories, 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.

A prominent approach to $SMT(\mathcal{T})$ which underlies several systems (e.g., CVCLITE [2], DLSAT [8], DPLL(T)/BarceLogic [10], MATHSAT [4], TSAT++ [1], ICS/YICES [9]), is based on extensions of SAT technology: a SAT engine is modified to enumerate boolean assignments, and integrated with a decision procedure for sets of literals in the theory $\mathcal{T}$ ($\mathcal{T}$*-solver*). The above schema is also followed to tackle the $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$ problem. The approach relies on a decision procedure able to decide the satisfiability of sets of literals in $\mathcal{T}_1 \cup \mathcal{T}_2$, that is typically based on an integration schema like Nelson-Oppen (NO) [11] (or its variant due to Shostak [13]): the $\mathcal{T}_i$*-solvers* are combined by means of a structured exchange of (disjunctions of) interface equalities (e_{ij}'s).

Unfortunately from a practical point of view this schema poses some challenges. First, the integration between the two $\mathcal{T}_i$-solvers is not trivial to implement. Second, the ability of $\mathcal{T}_i$-solvers of inferring (disjunctions of) interface equalities (hereafter e_{ij}-*deduction completeness*) required by NO is neither always easy to achieve nor always cheap to perform. (E.g., e_{ij}-deduction is cheap for $\mathcal{EUF}$ but can be very expensive for $\mathcal{LA}(\mathbb{Z})$.) Third, in case of non-convex theories (e.g., $\mathcal{LA}(\mathbb{Z})$), a backtrack search must be used to take care of the disjunctions that need to be managed.

In recent papers [3, 6] we have proposed a novel approach to $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$, called *Delayed Theory Combination* (DTC). The main idea is to avoid the integration schema between $\mathcal{T}_1$ and $\mathcal{T}_2$, and tighten the connection between each $\mathcal{T}_i$ and the SAT engine. While the truth assignment is being constructed, it is checked for consistency with respect to each theory in isolation. This can be seen as constructing two (possibly inconsistent) partial models for the original formula; the "merging" of the two partial models is enforced, on demand, since the solver is requested to find a complete assignment to the e_{ij}'s.

Compared to the NO schema, this approach has several advantages [3, 6]. First, it is easier to implement and analyze. Second, the approach does not rely on the $\mathcal{T}_i$*-solvers* being e_{ij}-deduction complete, although it can fully benefit from this property. Third, the DTC nicely encompasses the case of non-convex theories. On the negative side, in [3, 6] we noticed that these benefits are traded with a potential enlargement of the boolean search space which, in principle, could be up to exponential in the number of interface equalities generated. Thus, despite the positive empirical results presented in [3, 6], the latter fact represented, at least in theory, one possible drawback of DTC.

In this paper we show that this latter point was way too pessimistic. We present a comparative analysis of DTC vs. NO for $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$, and we introduce some novel theoretical results, for both convex and non-convex theories and for different deduction capabilities of the $\mathcal{T}$-solvers. These results show that, by exploiting the full power of advanced SAT techniques like backjumping and learning, DTC can be implemented in such a way as to mimic the behavior of NO, so that the amount of boolean branches required by DTC can be upper-bounded by the sum of the number of deductions and branches required by NO in order to perform the same tasks.

From these results we have that DTC generalizes NO, in the sense that:

- under the same hypotheses of e_{ij}-deduction-completeness of the $\mathcal{T}_i$*-solvers* required by NO, DTC emulates NO with no extra cost in terms of boolean search;
- in the more general case ($\mathcal{T}_i$*-solvers* with partial or no e_{ij}-deduction capabilities) DTC can mimic the behavior of NO, in such a way that all or part of the (pos-

sibly very expensive) e_{ij}-deductions are substituted with only few extra boolean branches.

We also notice that the capability of learning conflict clauses containing interface equalities, which is typical of DTC, allows for cutting branches corresponding to repeated deductions in an equivalent NO schema.

The paper is structured as follows. In Section 2 we present some background and introduce the Nelson-Oppen combination schema for $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$. DTC is then discussed in Section 3. We present our analysis in Sections 4 (where the case of e_{ij}-deduction completeness in the $\mathcal{T}_i$-solvers of DTC is examined) and 5 (where the $\mathcal{T}_i$-solvers employed by DTC are assumed to have limited or no deduction capabilities). Finally, in Section 6 we draw some conclusions.

For lack of space, the proofs of the theorems and a more detailed description of the algorithms are omitted here, and they are reported in an extended technical report [7].

2 SMT for combined theories via Nelson-Oppen's integration

2.1 Basic definitions and properties

Consider a theory $\mathcal{T}$ with equality. $\mathcal{T}$ 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 bit-vectors $\mathcal{BV}$ are typically not. In what follows, we shall assume to deal only with stably-infinite theories with equality and with disjoint signatures.

$\mathcal{T}$ is *convex* iff, for every collection $l_1, \ldots, l_k, e_1, \ldots, e_n$ of literals in $\mathcal{T}$ s.t. $e_1, \ldots, e_n$ are in the form $(x = y)$, x, y being variables, we have that

$$\{l_1, ..., l_k\} \models_{\mathcal{T}} \bigvee_{i=1}^{n} e_i \iff \{l_1, ..., l_k\} \models_{\mathcal{T}} e_i \text{ for some } 1 \leq i \leq n.$$

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.

Consider two theories $\mathcal{T}_1$, $\mathcal{T}_2$ with equality and disjoint signatures Σ_1, Σ_2. An atom ψ is i-pure if only $=$, variables and symbols from Σ_i occur in ψ. A formula φ is pure iff every atom in φ is i-pure for some $i \in \{1, 2\}$. Every non-pure $\mathcal{T}_1 \cup \mathcal{T}_2$ formula φ can be converted into an equivalently satisfiable pure formula φ' by recursively labeling terms t with fresh variables v_t, and by adding the atom $(v_t = t)$. E.g.:

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

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.

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 an 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}".

```
function Bool+T (φ: quantifier-free formula)
1       A^p ⟵ T2B(Atoms(φ))
2       φ^p ⟵ T2B(φ)
3       while Bool-satisfiable(φ^p) do
4           μ^p ⟵ pick_total_assign(A^p, φ^p)
5           (ρ, π) ⟵ T − satisfiable(B2T(μ^p))
6           if ρ = sat then return sat
7           φ^p ⟵ φ^p ∧ T2B(¬π)
8       end while
9       return unsat
end function
```

Fig. 1. A simplified view of enumeration-based T-satisfiability procedure: Bool+$\mathcal{T}$

Given a $\mathcal{T}$-inconsistent set of literals $L = \{l_1, \ldots, l_n\}$ in a theory $\mathcal{T}$, a *conflict set* η is an ($\mathcal{T}$-)inconsistent subset of L. η is *minimal* if none of its strict subsets is $\mathcal{T}$-inconsistent. We say that η is $\neg e_{ij}$*-minimal* iff $\eta \setminus \{\neg e_{ij}\}$ is no more $\mathcal{T}$-inconsistent, for every $\neg e_{ij} \in \eta$.

A $\mathcal{T}$*-solver* is a procedure that decides the consistency of an assignment μ in $\mathcal{T}$. An (propositional) assignment μ for a formula φ is a function $\mu : Atoms(\varphi) \mapsto \{true, false\}$. μ can be equivalently represented as a set of literals μ_S, where $\neg A \in \mu_S$ if $\mu(A) = false$, and $A \in \mu_S$ otherwise. μ can also equivalently be seen as a formula μ_φ, built as the conjunction of the literals in the set μ_S. (In the following, we will denote all such equivalent representations with μ. Moreover, we will denote with $\mu_{\mathcal{T}_i}$ the subassignment of μ containing only *i*-pure literals.) When a $\mathcal{T}$*-solver* detects the inconsistency of μ, it also returns a conflict set η of μ. Finally, we also require every $\mathcal{T}$*-solver* involved in either the NO schema or DTC to be *incremental* (it does not need to restart the computation from scratch to decide the satisfiability of μ' if it had already proved that of $\mu \subset \mu'$) and *backtrackable* (it can return to a previous state in an efficient manner) [11].

We say that a $\mathcal{T}$*-solver* is $\neg e_{ij}$*-minimal* (resp. *minimal*) if the conflict sets it returns are always $\neg e_{ij}$-minimal (resp. minimal). Notice that $\neg e_{ij}$-minimality is a much weaker requirement than minimality.

2.2 Satisfiability Modulo Theory

Fig. 1 presents Bool+$\mathcal{T}$, a (much simplified) decision procedure for $SMT(\mathcal{T})$. The function $Atoms(\varphi)$ takes a ground formula φ and returns the set of atoms which occur in φ. We use the notation φ^p to denote the *propositional abstraction* of φ, which is formed by the function $\mathcal{T}2\mathcal{B}$ that maps propositional variables to themselves, ground atoms into fresh propositional variables, and is homomorphic w.r.t. boolean operators and set inclusion. The function $\mathcal{B}2\mathcal{T}$ is the inverse of $\mathcal{T}2\mathcal{B}$. We use μ^p to denote a propositional assignment. (If $\mathcal{T}2\mathcal{B}(\mu) \models \mathcal{T}2\mathcal{B}(\varphi)$, then we say that μ *propositionally satisfies* φ.) The idea underlying the algorithm is that the truth assignments for the propositional abstraction of φ are enumerated and checked for satisfiability in $\mathcal{T}$. The procedure either returns sat if one such model is found, or returns unsat otherwise. The

function *pick_total_assign* returns a total assignment to the propositional variables in φ^p, that is, it assigns a truth value to all variables in $\mathcal{A}^p$. The function $\mathcal{T}$-satisfiable(μ) detects if the set of conjuncts μ is $\mathcal{T}$-satisfiable: if so, it returns (sat, $\emptyset$); otherwise, it returns (unsat, π), where $\pi \subseteq \mu$ is a $\mathcal{T}$-unsatisfiable set, called a *theory conflict set*. We call the negation of a conflict set, a *conflict clause*.

The algorithm is a coarse abstraction of the ones underlying TSAT++, MATHSAT, DLSAT, DPLL(T)/BarceLogic, CVCLITE, and ICS/YICES. The test for satisfiability and the extraction of the corresponding truth assignment are kept separate in this description only for the sake of simplicity.

In practice, the enumeration of truth assignments is carried out by means of efficient implementations of the DPLL algorithm [15], where a *partial assignment* μ^p is built incrementally, each time selecting an unassigned literal l (*literal selection*), called *decision literal*, according to some heuristic criterion, adding it to μ^p and performing all the other assignments which derive deterministically from this choice (*unit propagation*). When some assignment μ^p falsifies the formula returning a (boolean) conflict set π^p, or when $\mathcal{T}$-satisfiable$(\mathcal{B}2\mathcal{T}(\mu^p))$ fails returning a theory conflict set π, the negation $\neg\pi^p$ of (the boolean abstraction of) the conflict set is passed as a conflict clause to the boolean solver. Then $\neg\pi^p$ is added in conjunction to φ^p either temporarily or permanently (*learning*), and the algorithm backtracks up to the highest point in the search where a literal can be unit-propagated on $\neg\pi^p$ (*backjumping*). Learning also avoids generating the same conflicts in future branches.

An important variant [10] is that of building from $\neg\pi^p$ a "mixed boolean+theory conflict clause", by recursively removing non-decision literals l from the conflict clause by resolving the latter with the clause C_l which caused the unit-propagation of l; this is done until the conflict clause contains only decision literals (*last-UIP strategy*) or at most one non-decision literal assigned after the last decision (*first-UIP strategy*).[4]

Another important improvement is *early pruning (EP)*: before every literal selection, intermediate assignments are checked for $\mathcal{T}$-satisfiability and, if not $\mathcal{T}$-satisfiable, they are pruned (since no refinement can be $\mathcal{T}$-satisfiable). Finally, *theory deduction* can be used to reduce the search space by allowing the $\mathcal{T}$-solvers to explicitly return truth values for unassigned literals, which can be unit-propagated by the SAT solver. The interested reader is pointed to, e.g., [1, 4, 10, 5] for details and further references.

2.3 Nelson-Oppen's schema

Given two signature-disjoint stably infinite theories $\mathcal{T}_1$ and $\mathcal{T}_2$, the Nelson-Oppen combination schema [11], in the following referred to as NO, allows for solving the satisfiability problem for $\mathcal{T}_1 \cup \mathcal{T}_2$ (i.e. the problem of checking the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of sets of $\Sigma_1 \cup \Sigma_2$-literals) by using the satisfiability procedures for $\mathcal{T}_1$ and $\mathcal{T}_2$. The procedure is basically a structured interchange of information inferred from either theory and propagated to the other, until convergence is reached. The schema requires the exchange of information, the kind of which depends on the *convexity* of the involved theories. In the case of convex theories, the two solvers communicate to each other single interface

[4] These are standard techniques implemented in most SAT solvers in order to build the boolean conflict clauses [14].

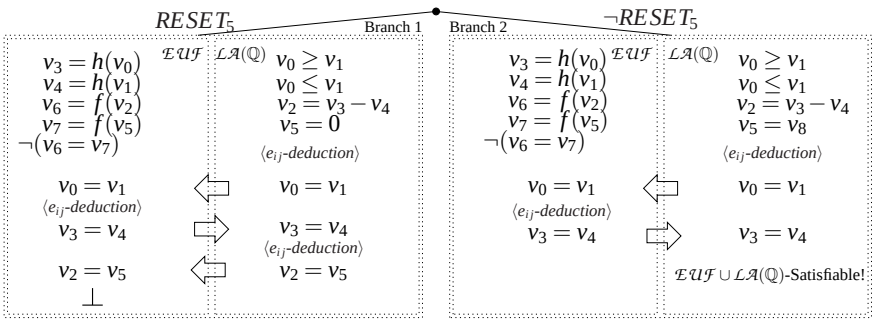

Fig. 2. Representation of the search tree for the formula of Example 1

equalities. In the case of non-convex theories, the NO schema becomes more complicated, because the two solvers need to exchange *arbitrary disjunctions of interface equalities*, which have to be managed within the decision procedure by means of case splitting and of *backtrack* search. In the latter case, the NO schema performs a number of *branches* to check the consistency of a set of literals 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.

Example 1 (convex case). Consider the following $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$ formula φ (cf Fig. 2)

$$\begin{array}{ll} \mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\ \mathcal{LA}(\mathbb{Q}): & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\ Both: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7). \end{array} \quad (1)$$

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.
After the first run of unit propagations, assume DPLL selects the literal $RESET_5$, resulting in the assignment

$$\begin{array}{l} \mu = \{\ (v_3 = h(v_0)), (v_4 = h(v_1)), (v_6 = f(v_2)), (v_7 = f(v_5)), (v_0 \geq v_1), \\ \qquad (v_0 \leq v_1), (v_2 = v_3 - v_4), \neg(v_6 = v_7), RESET_5, (v_5 = 0)\}, \end{array} \quad (2)$$

which propositionally satisfies φ. Now, the set of literals $\mu_{\mathcal{EUF}} \subset \mu$ is given to the $\mathcal{EUF}$ solver, which reports its consistency and deduces no new interface equality. Then the set $\mu_{\mathcal{LA}(\mathbb{Q})} \subset \mu$ 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 deduces the equality $(v_3 = v_4)$, which is in turn passed to the $\mathcal{LA}(\mathbb{Q})$ solver, that now as a consequence of this and the assignment $\mu_{\mathcal{LA}(\mathbb{Q})}$ 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 method reports the $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$-unsatisfiability

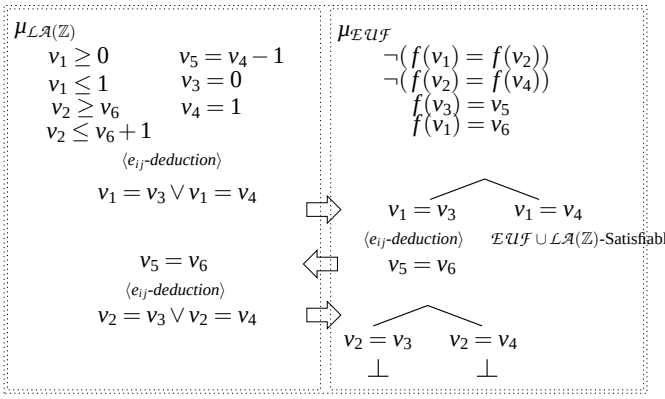

Fig. 3. Representation of the search tree for the formula of Example 2

of φ under the whole assignment μ. At this point, then, DPLL backtracks and tries assigning false to $RESET_5$, resulting in the new assignment

$$\mu = \{\ (v_3 = h(v_0)), (v_4 = h(v_1)), (v_6 = f(v_2)), (v_7 = f(v_5)), (v_0 \geq v_1), (v_0 \leq v_1),\\ (v_2 = v_3 - v_4), \neg(v_6 = v_7), \neg RESET_5, (v_5 = v_8))\},$$

which is found $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$-satisfiable (see Fig. 2). □

Example 2 (non-convex case). Consider the following $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Z})$ formula φ

$$\begin{array}{ll} \mathcal{EUF}: & \neg(f(v_1) = f(v_2)) \wedge \neg(f(v_2) = f(v_4)) \wedge (f(v_3) = v_5) \wedge (f(v_1) = v_6) \wedge \\ \mathcal{LA}(\mathbb{Z}): & (v_1 \geq 0) \wedge (v_1 \leq 1) \wedge (v_5 = v_4 - 1) \wedge (v_3 = 0) \wedge (v_4 = 1) \wedge \\ & (v_2 \geq v_6) \wedge (v_2 \leq v_6 + 1). \end{array} \quad (3)$$

Here (see Fig. 3) all the variables $(v_1, \ldots, v_6)$ are interface ones. φ 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 φ. The Nelson-Oppen combination schema then runs as follows. 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 the disjunction $(v_1 = v_3) \vee (v_1 = v_4)$. Next, there is a case-splitting and the two equalities $(v_1 = v_3)$ and $(v_1 = v_4)$ are passed to 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; at this point, the other case of the branch (i.e. the equality $(v_1 = v_4)$) is selected, and since the assignment

```
function Bool+T1+T2 (φi: quantifier-free formula)
1     φ ⟵ purify(φi)
2     A^p ⟵ T2B(Atoms(φ) ∪ interface_equalities(φ))
3     φ^p ⟵ T2B(φ)
4     while Bool-satisfiable (φ^p) do
5         μ1^p ∧ μ2^p ∧ μe^p = μ^p ⟵ pick_total_assign(A^p, φ^p)
6         (ρ1, π1) ⟵ T1-satisfiable (B2T(μ1^p ∧ μe^p))
7         (ρ2, π2) ⟵ T2-satisfiable (B2T(μ2^p ∧ μe^p))
8         if (ρ1 = sat ∧ ρ2 = sat) then return sat else
9             if ρ1 = unsat then φ^p ⟵ φ^p ∧ T2B(¬π1)
10            if ρ2 = unsat then φ^p ⟵ φ^p ∧ T2B(¬π2)
11    end while
12    return unsat
end function
```

Fig. 4. A simplified view of the Delayed Theory Combination procedure for $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$

$\mu_{\mathcal{EUF}} \cup \{(v_1 = v_4)\}$ is $\mathcal{EUF}$-consistent and no new interface equality is deduced, the Nelson-Oppen method reports the $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Z})$-satisfiability of φ under the whole assignment μ. □

3 SMT for combined theories via Delayed Theory Combination

In the Delayed Theory Combination (DTC) schema [3, 6], the $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$ problem is tackled in a different way: each of the two $\mathcal{T}_i$ solvers works in isolation, without direct exchange of information. Their mutual consistency is ensured by augmenting the input problem with all interface equalities e_{ij}, even if these do not occur in the original problem. The enumeration of assignments includes not only the atoms in the formula, but also the interface equalities e_{ij}. Both theory solvers receive, from the boolean level, the same truth assignment μ_e for e_{ij}: under such conditions, the two "partial" models found by each decision procedure can be merged into a model for the input formula.

A simplified view of the algorithm is presented in Fig. 4. Initially (lines 1–3), the formula is purified, the *new* e_{ij}'s are created and added to the set of propositional symbols $\mathcal{A}^p$, and the propositional abstraction φ^p of φ is created. Then, the main loop is entered (lines 4–11): while φ^p is propositionally satisfiable (line 4), a satisfying truth assignment μ^p is selected (line 5). It is important to stress that truth values are associated not only to atoms in φ, but also to the e_{ij} atoms, even though they do not occur in φ. μ^p is then (implicitly) separated into $\mu_1^p \wedge \mu_e^p \wedge \mu_2^p$, where $\mathcal{B}2\mathcal{T}(\mu_i^p)$ is a set of i-pure literals and $\mathcal{B}2\mathcal{T}(\mu_e^p)$ is a set of e_{ij}-literals. The relevant parts of μ^p are checked for consistency against each theory (lines 6–7); $\mathcal{T}_i$-satisfiable(μ) returns a pair (ρ_i, π_i), where ρ_i is unsat iff μ is unsatisfiable in $\mathcal{T}_i$, and sat otherwise. If both calls to $\mathcal{T}_i$-satisfiable return sat, then the formula is satisfiable. Otherwise, when ρ_i is unsat, then π_i is a theory conflict set, i.e. $\pi_i \subseteq \mu$ and π_i is $\mathcal{T}_i$-unsatisfiable. Then, φ^p is strengthened to exclude truth assignments which may fail in the same way (line 9–10), and the loop is resumed.

Unsatisfiability is returned (line 12) when the loop is exited without having found a model.

In practical implementations of DTC, the search for a satisfactory assignment is based on a modern DPLL engine, performing literal selection, unit-propagation, backjumping and learning, early pruning, and theory deduction, as explained in §2.2. In particular, DTC can be enhanced by e_{ij}-deduction, in which e_{ij}'s can by deduced by the $\mathcal{T}_i$-*solvers* and hence unit-propagated. We refer the reader to [3, 6] for a more detailed discussion.

Notation-wise, we call "*new*" e_{ij}'s all the interface equalities e_{ij}'s which do not occur in any clause of the input formula φ (including all the clauses learned). Moreover, we often write sets of literals $\{l_1,...,l_n\}$ as conjunctions $l_1 \wedge ... \wedge l_n$, and we often write clauses $(\bigvee_i l_i) \vee (\bigvee_j l_j)$ as implications: $(\bigwedge_i \neg l_i) \rightarrow (\bigvee_j l_j)$ or $(\bigwedge_i \neg l_i \wedge \bigwedge_j \neg l_j) \rightarrow \bot$.

Hereafter, for the sake of proving the theoretical results in §4 and §5, we assume that DTC implements the following strategy.

Strategy 1 (NO emulation)

1. *All the conflict clauses derived by theory conflicts are learned.* [5]
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 §2.2.* [6]
3. *The literal selection heuristic and the $\mathcal{T}_i$-solvers calls are such that:*
   (i) *new e_{ij}'s are selected only after all the other literals have been assigned,*
   (ii) *Early pruning (EP) is applied before every selection of a new e_{ij},* [7]
   (iii) *the new e_{ij}'s selected are always assigned false,*
   (iv) *each $\mathcal{T}_i$-solver is invoked only if at least one literal (which has not been deduced singularly by $\mathcal{T}_i$-solver itself) has been added to its input since the last call.* [8]
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 $\mathcal{T}2\mathcal{B}(\mu^* \rightarrow \bigvee_{j=1}^{k} e_j)$ 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] That is, if one $\mathcal{T}_i$-*solver* returns a conflict set π, then the conflict clauses $\mathcal{T}2\mathcal{B}(\neg\pi)$ is always added to φ^p, either temporarily or permanently.

[6] That is, each conflict clause contains all and only (the negation of) the decision literals which forced the unit-propagation or the e_{ij}-deduction of those in the theory conflict.

[7] That is, before adding a new (negated) e_{ij} to μ, the $\mathcal{T}_i$-satisfiability of μ is checked for both $\mathcal{T}_i$'s by calling the $\mathcal{T}_i$-*solver*'s. If μ is found $\mathcal{T}_i$-inconsistent for some $\mathcal{T}_i$, then the procedure backtracks.

[8] 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 $\mu \cup \{e_{ij}\}$.

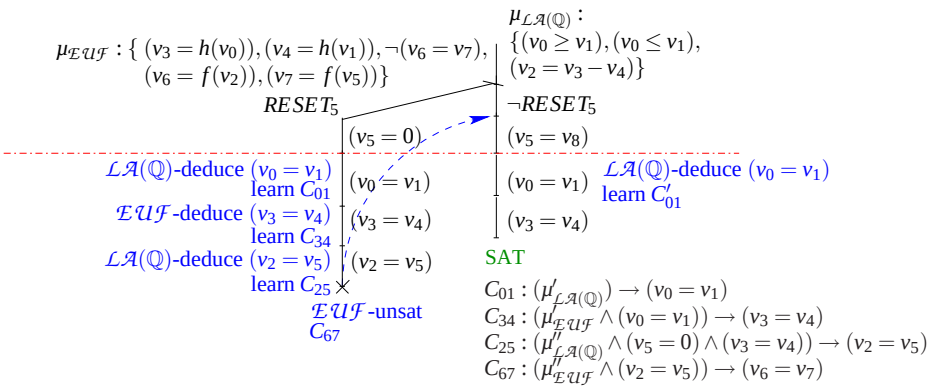

Fig. 5. DTC execution of Example 3 on $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{EUF}$, with e_{ij}-deduction-complete $\mathcal{T}_i$-solvers.

5. **[If and only if both $\mathcal{T}_i$-*solvers* are e_{ij}-deduction complete]**
If a total 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".* [9]

4 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 1. *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 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 be the number of boolean branches required in the* DTC *proof. Then we have:*

$$dtc\_br \leq no\_br \tag{4}$$

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.

Theorem 1 states that, under the same hypotheses of e_{ij}-deduction as NO, DTC emulates NO with no extra cost in terms of boolean search.

Example 3 (convex case). Consider again the $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$ formula φ of Example 1. Figure 5 illustrates a DTC execution when both $\mathcal{T}_i$-solvers are e_{ij}-deduction complete.

On the left branch (when $RESET_5$ is selected), after the unit-propagation of $(v_5 = 0)$, the $\mathcal{LA}(\mathbb{Q})$ solver deduces $(v_0 = v_1)$, and thus by Step 4. (i) of Strategy 1, the clause C_{01} is learned and $(v_0 = v_1)$ is unit-propagated. As a consequence of this, the $\mathcal{EUF}$

[9] Step 5. is identical to the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability termination condition of NO.

solver can deduce $(v_3 = v_4)$, resulting in 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 $(v_2 = v_5)$, with the resulting learning of C_{25} and unit-propagation of the deduced equality.

At this point, $\mu''_{\mathcal{EUF}} \cup \{(v_2 = v_5)\}$ [10] 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}, $\neg(v_6 = v_7)$, and $(RESET_5 \rightarrow (v_5 = 0))$ as explained in Step 2. of Strategy 1, obtaining a mixed theory+boolean conflict clause C'_{67} in the form $(\mu^* \wedge RESET_5) \rightarrow \bot$ s.t. μ^* contains no interface equality. C'_{67} forces DTC to backjump up to the last branching point. Then the execution of the right branch begins with the unit-propagation of $\neg RESET_5$ on C'_{67} and hence of $(v_5 = v_8)$ on $\neg RESET_5 \rightarrow (v_5 = v_8)$, which produces an assignment propositionally satisfying φ. The theory solvers are invoked, and the $\mathcal{LA}(\mathbb{Q})$ solver deduces again $(v_0 = v_1)$, learning a clause C'_{01} which is similar to C_{01} except for the fact that it may contain the redundant literal $(v_5 = v_8)$ instead of $(v_5 = 0)$. [11] Then $(v_3 = v_4)$ is unit-propagated on C_{34}. At this point, since both theory solvers cannot deduce any new e_{ij}, by Step 5. of Strategy 1 DTC concludes that φ is $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$-satisfiable. □

Notice that the left branch of the DTC search tree of Figure 5 mimics directly that of the NO execution of Figure 2. 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*, but rather they are added to the current assignment μ and unit-propagated.

In the right branch, instead, all values are assigned directly by unit-propagation. This fact illustrates one further potential advantage of DTC with respect to NO: the fact that new e_{ij}'s are known a priori to the DPLL engine allows their inclusion in the learned clauses derived by theory conflicts. Thanks to unit-propagation, this makes it possible to assign truth values to them *directly at the boolean level*, without performing the (potentially costly) invocation of the $\mathcal{T}_i$-solvers. In the traditional NO schema, this fact does not come naturally, because the boolean solver knows nothing about the e_{ij}'s.

We consider now the case where some $\mathcal{T}_i$'s are non-convex.

Example 4 (non-convex case). Consider the $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Z})$ formula φ and assignment μ of Example 2. Figure 6 illustrates a DTC execution when both $\mathcal{T}_i$-solvers are e_{ij}-deduction complete.

The first invocation of the $\mathcal{LA}(\mathbb{Z})$ solver results in deducing of the disjunction $(v_1 = v_4) \vee (v_1 = v_3)$ and learning of the corresponding clause 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

[10] Hereafter, $\mu'_{\mathcal{T}}$, $\mu''_{\mathcal{T}}$, $\mu'''_{\mathcal{T}}$ will denote generic subsets of $\mu_{\mathcal{T}}$, $\mathcal{T} \in \{\mathcal{EUF}, \mathcal{LA}(\mathbb{Q}), \mathcal{LA}(\mathbb{Z})\}$.

[11] Here we assume the "worst" case in which $\mu'_{\mathcal{LA}(\mathbb{Q})}$ in C_{01} contains the (redundant) literal $(v_5 = 0)$. If this is not the case, then $(v_0 = v_1)$ is directly unit-propagated on C_{01}, without calling the theory solvers.

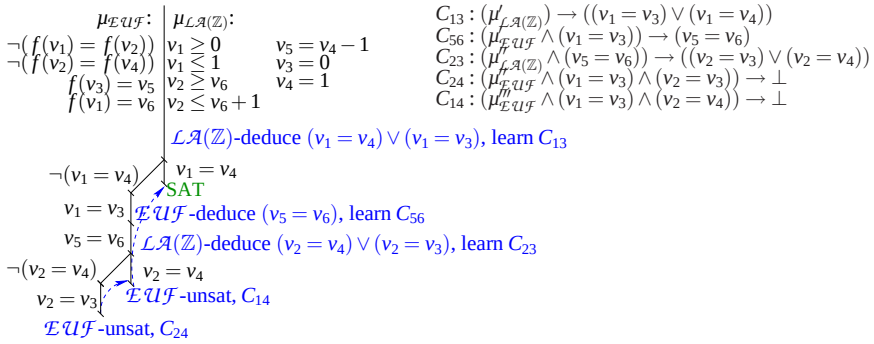

Fig. 6. DTC execution of Ex 4 on $\mathcal{LA}(\mathbb{Z}) \cup \mathcal{EUF}$, with e_{ij}-deduction-complete $\mathcal{T}_i$-solvers.

(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. □

As with the convex example, notice that the DTC search tree of Figure 6 mimics directly that of the NO execution of Figure 3 (both *dtc_br* and *no_br* are equal to 3.)

5 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:*

$$dtc\_br + dtc\_ded \leq no\_br + no\_ded, \tag{5}$$

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.

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

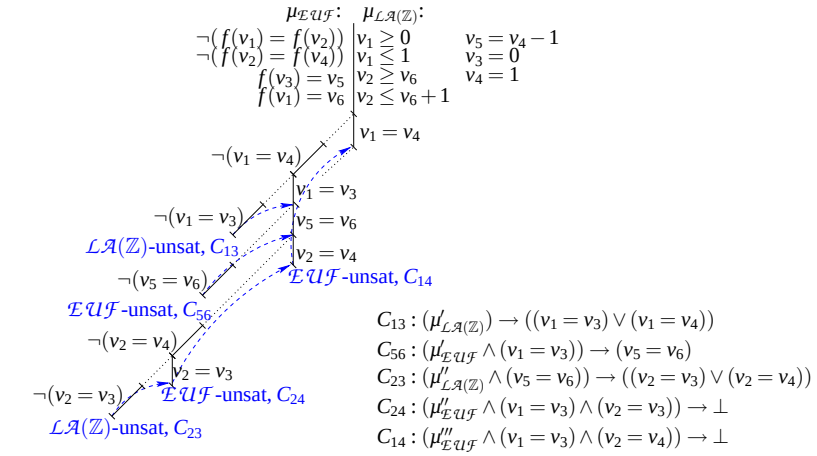

Fig. 7. DTC execution of Example 5 on $\mathcal{LA}(\mathbb{Z}) \cup \mathcal{EUF}$, with no e_{ij}-deduction. The clauses C_{ij}'s are the same as those of Fig. 6.

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 fails and generates a conflict set π, it backjumps up to the second-most-recently-assigned $\neg e_{ij}$ in π, if any [7]. (See, e.g., the case of C_{23} in Figure 7.) 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 5 (no e_{ij}-deduction, non-convex case). Consider the $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Z})$ formula φ (3) and the assignment μ of Example 2. Look at Fig. 7. Both $\mu_{\mathcal{LA}(\mathbb{Z})}$ and $\mu_{\mathcal{EUF}}$ are found consistent in the respective theories by the respective solvers.

Then DTC starts selecting new $\neg e_{ij}$'s, and proceeds without causing conflicts, 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 $\mu'_{\mathcal{LA}(\mathbb{Z})} \cup \{\neg(v_1 = v_3), \neg(v_1 = v_4)\}$. Thus DTC learns the corresponding clause C_{13} (see Fig 7) and backjumps up to the highest point which allows for unit-propagating $(v_1 = v_3)$ on C_{13}, and performs such unit propagation. Then DTC starts and 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 DTC backtracks over the last chain of $\neg e_{ij}$'s and unit-propagates $(v_5 = v_6)$.

Again, DTC selects a chain of new $\neg e_{ij}$'s without causing conflicts, 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, learning the 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 $(v_2 = v_3)$), DTC obtains a clause C'_{24} : $(\mu''_{\mathcal{LA}(\mathbb{Z})} \wedge \mu''_{\mathcal{EUF}} \wedge (v_5 = v_6) \wedge (v_1 = v_3)) \rightarrow (v_2 = v_4)$, which allows it for backjumping 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}. 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), obtaining the clause C'_{14} : $(\mu'_{\mathcal{LA}(\mathbb{Z})} \wedge \mu''_{\mathcal{LA}(\mathbb{Z})} \wedge \mu'_{\mathcal{EUF}} \wedge \mu''_{\mathcal{EUF}} \wedge \mu'''_{\mathcal{EUF}}) \rightarrow (v_1 = v_4)$, which allows for backjumping up to μ and unit-propagating $(v_1 = v_4)$.

Finally, DTC 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.

Comparing with Fig. 3, $dtc\_br = 6$, $dtc\_ded = 0$, $no\_ded = 3$ and $no\_br = 3$. □

Notice that the three leftmost diagonal branches in Fig. 7 obtain the same effect as the e_{ij}-deduction steps in Fig. 6 (and in Fig. 3).

6 Conclusions

Theorem 1 shows that, under the same hypotheses of e_{ij}-deduction-completeness as NO, DTC can emulate NO, with no extra boolean search. Theorem 2 shows that, under the hypothesis of $\neg e_{ij}$-minimality, even $\mathcal{T}_i$-solvers with limited or no e_{ij}-deduction capabilities allow DTC to emulate NO, at the cost of (at most) one extra boolean branch for every (possibly very expensive) e_{ij}-deduction performed by NO. Both results also highlight the fact that DTC naturally allows for learning clauses containing e_{ij}'s, which can be used in subsequent branches to prune search and avoid redoing the same search/deductions from scratch.

We remark that Strategy 1 has been conceived only for mimicking NO, and by no means it is assumed to be the most efficient strategy for DTC. (E.g., Step 3.(ii) can be substituted with a weakened version of EP [4], and more efficient literal selection strategies might be preferable to Step 3.(i) and (iii).) Some alternatives are currently under investigation, and their theoretical properties and practical performance are subjects for future work.

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 [12] and [4] respectively.) For other theories, the problem of $\neg e_{ij}$-minimization opens a novel research branch. [12] 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.

It is also important to notice that, in general, only a fraction of the assignments μ enumerated turn out to be $\mathcal{T}_i$-satisfiable for both $\mathcal{T}_i$'s, so that to require the boolean

[12] 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.

search on the e_{ij}'s. Thus, for all the other branches, DTC may save the effort of many failed attempts of deducing implied e_{ij}'s.

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., $\mathcal{EUF}$ [12]), 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.

References

1. A. Armando, C. Castellini, E. Giunchiglia, and M. Maratea. A SAT-based Decision Procedure for the Boolean Combination of Difference Constraints. In *Proc. SAT'04*, 2004.
2. C.L. Barrett and S. Berezin. CVC Lite: A New Implementation of the Cooperating Validity Checker. In *Proc. CAV'04*, volume 3114 of *LNCS*. Springer, 2004.
3. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P.van Rossum, S. Ranise, and R. Sebastiani. Efficient Satisfiability Modulo Theories via Delayed Theory Combination. In *Proc. Int. Conf. on Computer-Aided Verification, CAV 2005.*, volume 3576 of *LNCS*. Springer, 2005.
4. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P.van Rossum, S. Schulz, and R. Sebastiani. An incremental and Layered Procedure for the Satisfiability of Linear Arithmetic Logic. In *Proc. TACAS'05*, volume 3440 of *LNCS*. Springer, 2005.
5. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P.van Rossum, S. Schulz, and R. Sebastiani. MathSAT: A Tight Integration of SAT and Mathematical Decision Procedure. *Journal of Automated Reasoning*, 2005. to appear.
6. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani. Efficient Theory Combination via Boolean Search. *Information and Computation*, 2005. To appear.
7. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. Delayed Theory Combination vs. Nelson-Oppen for Satisfiability Modulo Theories: a Comparative Analysis. Technical Report DIT-06-032, DIT, University of Trento, 2006. Available at `http://dit.unitn.it/~rseba/papers/lpar06_dtc_extended.pdf`.
8. S. Cotton, E. Asarin, O. Maler, and P. Niebert. Some Progress in Satisfiability Checking for Difference Logic. In *Proc. FORMATS-FTRTFT 2004*, 2004.
9. J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated Canonizer and Solver. In *Proc. CAV'01*, volume 2102 of *LNCS*, pages 246–249, 2001.
10. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *Proc. CAV'04*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004.
11. G. Nelson and D.C. Oppen. Simplification by Cooperating Decision Procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, 1979.
12. R. Nieuwenhuis and A. Oliveras. Congruence Closure with Integer Offsets. In *Proc. 10th LPAR*, number 2850 in LNAI, pages 77–89. Springer, 2003.
13. R.E. Shostak. Deciding Combinations of Theories. *Journal of the ACM*, 31:1–12, 1984.
14. L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proc. ICCAD '01*. IEEE Press, 2001.
15. L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *Proc. CAV'02*, number 2404 in LNCS, pages 17–36. Springer, 2002.