Available online at www.sciencedirect.com

ScienceDirect

ELSEVIER

Information and Computation 204 (2006) 1493–1525

Information and Computation

www.elsevier.com/locate/ic

Efficient theory combination via boolean search[☆]

Marco Bozzano [a], Roberto Bruttomesso [a], Alessandro Cimatti [a,*], Tommi Junttila [b], Silvio Ranise [c], Peter van Rossum [d], Roberto Sebastiani [e]

[a] *ITC-IRST, Via Sommarive 18, 38050 Trento, Italy*
[b] *Helsinki University of Technology, P.O.Box 5400, FI-02015 TKK, Finland*
[c] *LORIA and INRIA-Lorraine, 615, rue du Jardin Botanique, BP 101, 54602 Villers les Nancy, Cedex, France*
[d] *Radboud University Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands*
[e] *DIT, Università di Trento, Via Sommarive 14, 38050 Trento, Italy*

Received 1 January 2005

Abstract

Many approaches to deciding the satisfiability of quantifier-free formulae with respect to a background theory T—also known as Satisfiability Modulo Theory, or SMT(T)—rely on the integration between an enumerator of truth assignments and a decision procedure for conjunction of literals in T. When the background theory T is the combination $T_1 \cup T_2$ of two simpler theories, the approach is typically instantiated by means of a theory combination schema (e.g. Nelson–Oppen, Shostak). In this paper we propose a new approach to SMT($T_1 \cup T_2$), where the enumerator of truth assignments is integrated with two decision procedures, one for T_1 and one for T_2, acting independently from each other. The key idea is to search for a truth assignment not only to the atoms occurring in the formula, but also to all the equalities between variables which are shared between the theories. This approach is simple and expressive: for instance, no modification is required to handle non-convex theories (as opposed to traditional Nelson–Oppen combinations which require a mechanism for splitting). Furthermore, it can be made practical by leveraging on state-of-the-art boolean and SMT

[☆] 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. The work of T. Junttila has also been supported by the Academy of Finland, projects 53695 and 211025.

[*] Corresponding author. Fax: +39 0461 314 591.

E-mail addresses: bozzano@itc.it (M. Bozzano), bruttomesso@itc.it (R. Bruttomesso), cimatti@itc.it (A. Cimatti), Tommi.Junttila@tkk.fi (T. Junttila), Silvio.Ranise@loria.fr (S. Ranise), petervr@sci.kun.nl (P. van Rossum), roberto.sebastiani@unitn.it (R. Sebastiani).

0890-5401/$ - see front matter © 2006 Published by Elsevier Inc.
doi:10.1016/j.ic.2005.05.011

search techniques, and on theory layering (i.e., cheaper reasoning first, and more often). We provide thorough experimental evidence to support our claims: we instantiate the framework with two decision procedures for the combinations of Equality and Uninterpreted Functions ($\mathcal{EUF}$) and Linear Arithmetic ($\mathcal{LA}$), both for (the convex case of) reals and for (the non-convex case of) integers; we analyze the impact of the different optimizations on a variety of test cases; and we compare the approach with state-of-the-art competitor tools, showing that our implemented tool compares positively with them, sometimes with dramatic gains in performance.

1. Introduction

The problem of deciding the satisfiability of a quantifier-free formula with respect to a background theory T, also known under the name of Satisfiability Modulo Theories (*SMT*), is being recognized as increasingly important due to its application to the domain of verification. Notable theories of interest are Equality and Uninterpreted Functions ($\mathcal{EUF}$), Linear Arithmetic ($\mathcal{LA}$), both over the reals ($\mathcal{LA}(Rat)$) and the integers ($\mathcal{LA}(Int)$), and its subclass of Difference Logics ($\mathcal{DL}$). In fact, representation capabilities beyond propositional logic allow for a natural modeling of a number of real-world problems, e.g., verification of pipelines, equivalence checking of circuits at Register-Transfer Level (RTL), discharge of proof obligations in software systems, model checking real-time embedded systems. Particularly relevant is the case of $SMT(T_1 \cup T_2)$, where the background theory T is the combination of two (or more) simpler theories T_1 and T_2. This is because it is often the case that different kinds of information has to be taken into account. For instance, RTL circuits may exhibit hierarchical structure, memory accesses, and arithmetic. Similarly, the analysis of data-intensive reactive programs (e.g. LUSTRE, Signal) may greatly benefit from the ability to abstract foreign functions. Being able to decide such combinations of theories automatically avoids the need for manual breakdown of verification problems into subparts, and results in augmented capacity of verification tools, and higher design productivity.

A prominent approach to $SMT(T)$, which underlies several verification tools (e.g., MathSAT [5], DLSAT [9], DPLL(T) [17], TSAT++ [1], ICS [16], CVCLite [3], haRVey [12], Verifun [15], Zapato [6]), is based on extensions of propositional SAT technology: a SAT engine is modified to enumerate boolean assignments, and integrated with a decision procedure for the theory T. This schema, denoted as $\mathsf{Bool}+T$ in the following, is also followed to tackle the $SMT(T_1 \cup T_2)$ problem. The approach relies on a decision procedure able to decide the satisfiability of conjunctions of literals in $T_1 \cup T_2$, that is typically based on an integration schema like Nelson–Oppen (N.O.) [23] or Shostak [30]: decision procedures for each T_i are combined by means of a structured exchange of so-called interface formulae. In particular, in the case of convex theories, interface equalities (that is, equalities between variables which are shared between two theories) are exchanged; in the case of non-convex theories, disjunctions of interface equalities have to be managed, and a case-split mechanism is also required. The resulting procedure for $SMT(T_1 \cup T_2)$ is denoted as $\mathsf{Bool}+\mathsf{no}(T_1, T_2)$ in the following. We remark that the problem of $SMT(T_1 \cup T_2)$ is far from trivial: the quest for efficiency clashes with the increased complexity in the decision procedure, also due to the intricacies of the combination schema.

In this paper, we tackle the $SMT(T_1 \cup T_2)$ problem by deviating from the $\mathsf{Bool}+\mathsf{no}(T_1, T_2)$ approach, and avoiding the need for an integration schema between T_1 and T_2. Rather, we tighten the connection between each T_i and the boolean level: while the truth assignment is being constructed, each theory is checked for consistency in isolation. This can be seen as constructing two partial models for the original formula, without the guarantee that they will be mutually consistent; the "merging" of the two partial models is enforced, on demand, since the solver is requested to find a truth assignment to all the possible equalities between the interface variables (i.e., variables belonging both to T_1 and T_2).

This approach is very general and expressive. It nicely encompasses the case of non-convex theories, whereas in the $\mathsf{no}(T_1, T_2)$ case, a backtrack search is used to take care of the disjunctions that need to be managed. Furthermore, it is very simple to understand and analyze. In fact, its correctness can be easily stated by generalizing (to the case of formulae with arbitrary boolean structure) the proof of correctness for the nondeterministic variant of the Nelson–Oppen integration schema [36].

The approach is also amenable to several implementation advantages. First, each of the solvers can be implemented and optimized without taking into account the others; for instance, when the problem falls within one T_i, the solver behaves exactly as $\mathsf{Bool}+T$. Second, the approach does not rely on the solvers being deduction-complete. This enables us to explore the trade-off between which deduction is beneficial to efficiency and which is in fact hampering the search—or too difficult to implement. Furthermore, it is possible to exploit boolean search optimizations and *SMT* techniques to optimize the management of interface equalities.

We provide thorough experimental evidence to support our claims: we instantiate the framework for the combination of $\mathcal{EUF}$ and $\mathcal{LA}$, both for (the convex case of) reals and for (the non-convex case of) integers; we analyze the impact of the different optimizations on a variety of test cases; and we compare the approach with state-of-the-art competitor tools, showing that our implemented tool compares positively with them, sometimes with dramatic gains in performance.

This paper is structured as follows. We first present some background on the integration of theories, and on the Nelson–Oppen combination schema (Section 2), and describe the $\mathsf{Bool}+T$ decision procedure (Section 3). In Section 4 we present the $\mathsf{Bool}+T_1+T_2$ procedure, discuss its features, and prove its formal properties. In Section 5 we describe the architectural choices underlying the instantiation of the $\mathsf{Bool}+T_1+T_2$ in MathSAT. In Section 6 we discuss related work, and in Section 7 we experimentally evaluate our approach. In Section 8 we draw some conclusions and outline some directions for future work. Some of the material presented in this paper already appeared in [4].

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 T_1 and T_2. The generalization to more than two theories is straightforward.

2. Preliminary notions

We assume the usual syntactic notions of first-order logic with equality as defined, e.g., in [11]. Let Σ be a first-order signature containing function and predicate symbols with their arities and $\mathcal{X}$ be a set of variables. A 0-ary function symbol is called a *constant*. A Σ-*term* is a first-order term built out of the function symbols in Σ and the variables in $\mathcal{X}$. We use the standard notion of substitution. We write substitution applications in postfix notation, e.g. $t\sigma$ for a term t and a substitution σ. The set of variables occurring in a term t is denoted by $Var(t)$. If $t_1, \ldots, t_n$ are Σ-terms and p is a predicate symbol, then $p(t_1, \ldots, t_n)$ is a Σ-*atom*. If l and r are two Σ-terms, then the Σ-atom $l = r$ is called a Σ-*equality* and $\neg(l = r)$ (also written as $l \neq r$) is called a Σ-*disequality*. A Σ-*formula* ϕ is built in the usual way out of the universal and existential quantifiers, boolean connectives, and Σ-atoms. The set of Σ-atoms occurring in ϕ is denoted by $Atoms(\phi)$. A Σ-*literal* is either a Σ-atom or its negation. If φ is a formula, then $Var(\varphi)$ denotes the set of free variables in φ. We call a formula *quantifier-free* if it does not contain quantifiers, and a *sentence* if it has no free variables. Substitution applications are extended to arbitrary first-order formulas, and are written in postfix notation, e.g. $\varphi\sigma$ for a formula φ and a substitution σ. An *identification* over a set $\mathcal{V}$ of variables is an idempotent substitution from $\mathcal{V}$ to $\mathcal{V}$. Any identification σ over a set $\mathcal{V}$ of variables defines a partition of $\mathcal{V}$ and identifies all the variables in the same equivalence class of the partition with a representative of that class. If σ is an identification over a set $\mathcal{V}$ of variables and $\bowtie \in \{=, \neq\}$, then $\widehat{\sigma_{\bowtie}}$ denotes the conjunction

$$\bigwedge_{\{(x,y) | x\sigma \bowtie y\sigma \text{ and } x,y \in \mathcal{V}\}} x \bowtie y$$

of literals. Notice that $\widehat{\sigma_{=}}$ expresses the fact that any two variables identified by an identification σ must take identical value while $\widehat{\sigma_{\neq}}$ expresses the fact that any two variables not identified by σ must take distinct value. So, the formula $\widehat{\sigma_{=}} \wedge \widehat{\sigma_{\neq}}$ (abbreviated with $\widehat{\sigma}$ below) faithfully represents the identification σ over the set $\mathcal{V}$ of variables.

We also assume the usual first-order notions of interpretation, satisfiability, validity, logical consequence, and theory, as given, e.g., in [13]. We write $\Gamma \models \phi$ to denote that the formula ϕ is a logical consequence of the (possibly infinite) set Γ of formulae. A *first-order theory* is a set of first-order sentences. A Σ-*theory* is a theory all of whose sentences have signature Σ. All the theories we consider are first-order theories *with equality*, which means that the equality symbol $=$ is a predefined logical constant and it is always interpreted as a relation which is reflexive, symmetric, transitive, and it is also a congruence. Since the equality symbol is a predefined logical constant, it will not be included in any signature Σ considered in this paper (this is an important technical detail to precisely state the results about the combination of theories below). A Σ-structure $\mathcal{A}$ is a model of a Σ-theory T if $\mathcal{A}$ satisfies every sentence in T. A Σ-formula is *satisfiable in T* (or T-*satisfiable*) if it is satisfiable in a model of T. Two Σ-formulas φ and ψ are *equisatisfiable in T* if φ is satisfiable in T iff ψ is satisfiable in T. The *satisfiability problem* for a theory T amounts to establishing whether any given finite quantifier-free conjunction of literals (or equivalently, any given finite set of literals) is T-satisfiable or not. A *satisfiability procedure* for T is any algorithm that solves the satisfiability problem for T. The satisfiability of any quantifier-free formula can be reduced to the satisfiability of sets of literals by converting to disjunctive normal form (DNF) and then splitting on disjunctions, e.g., checking whether $S_1 \vee S_2$ (where S_1 and S_2 are conjunction of literals) is T-satisfiable reduces to checking whether either S_1 or S_2 is T-satisfiable. Indeed, the conversion to DNF may result in an exponential blow-up of the size of the formula. A much more efficient way to tackle this problem in practice is described in Section 3. Notice that the problem of checking the T-satisfiability of quantifier-free formulae is NP-hard (since it subsumes the problem of checking the satisfiability of boolean formulae).

2.1. Semantic properties of theories

In the sequel, let Σ_1 and Σ_2 be two disjoint signatures (i.e. $\Sigma_1 \cap \Sigma_2 = \emptyset$), and let T_i be a Σ_i-theory for $i = 1, 2$.

Definition 1. A Σ-theory T is *stably infinite* if for each T-satisfiable Σ-formula φ, there exists a model of T whose domain is infinite and which satisfies φ.

Definition 2. A *Nelson–Oppen* theory is a stably infinite theory that admits a satisfiability algorithm.

Definition 3. A conjunction Γ of Σ-literals is *convex* in a Σ-theory T if for each disjunction $\bigvee_{i=1}^{n} x_i = y_i$ (where x_i, y_i are variables and $i = 1, ..., n$) we have that $T \cup \Gamma \models \bigvee_{i=1}^{n} x_i = y_i$ iff $T \cup \Gamma \models x_i = y_i$ for some $i \in \{1, ..., n\}$. A Σ-theory T is *convex* iff all the conjunctions of Σ-literals are convex in T.

Notice that any convex theory whose models are non-trivial (i.e. the domains of the models have all cardinality strictly greater than one) is stably infinite [7].

In this paper, we will consider the following three theories (however, the proposed approach is not limited to these):

- The theory $\mathcal{EUF}$ of equality whose signature contains a finite set of uninterpreted function and constant symbols, and such that the equality symbol $=$ is interpreted as the equality relation.
- The quantifier-free fragment of Linear Arithmetic either over the rationals, denoted by $\mathcal{LA}(Rat)$, or over the integers, denoted by $\mathcal{LA}(Int)$. The signatures of $\mathcal{LA}(Rat)$ and $\mathcal{LA}(Int)$ contain the constants 0 and 1, the binary $+$ symbol, the unary $-$ symbol, and the predicate symbol $\leqslant$ which all have the usual arithmetic meaning. We also abbreviate with n the term $1 + 1 + \cdots + 1$ containing n occurrences of the constant 1, and with $n \cdot x$ the term $x + x + \cdots + x$, x being a variable.

All three theories are stably infinite but only $\mathcal{E}$ and $\mathcal{LA}(Rat)$ are convex, while $\mathcal{LA}(Int)$ is non-convex. To see that $\mathcal{LA}(Int)$ is non-convex, it is sufficient to notice that the set $\{x_1 = 1, x_2 = 2, x_1 \leqslant x, x \leqslant x_2\}$ entails $x = x_1 \vee x = x_2$ but neither $x = x_1$ nor $x = x_2$, where x, x_1, x_2 are variables.

2.2. Theory combination

In order to describe and prove the correctness of our approach for solving the satisfiability problem for a combination of theories, we need some additional notation and one Lemma. These are also used in the context of the Nelson–Oppen combination schema.

A $\Sigma_1 \cup \Sigma_2$-term t is an *i-term* if it is a variable or if it has the form $f(t_1, ..., t_n)$, where f is in Σ_i (for $i = 1, 2$ and $n \geqslant 0$). 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 formula is said to be *pure* if every atom occurring in the formula is i-pure for some $i \in \{1, 2\}$.

First we need to solve the following problem: we consider $\Sigma_1 \cup \Sigma_2$-literals while the satisfiability procedure for a theory T_i only handles Σ_i-pure literals. The standard solution consists of *purifying* any conjunction Φ of $\Sigma_1 \cup \Sigma_2$-literals into a conjunction $\Phi_1 \wedge \Phi_2$ where Φ_i is a conjunction of i-pure literals (for $i = 1, 2$). This is achieved by replacing each alien subterm t by a new variable x and adding the equality $x = t$ to Φ. Obviously, the purification process always terminates (since there are only finitely many alien sub-terms in Φ), yielding $\Phi_1 \wedge \Phi_2$ such that $\Phi_1 \wedge \Phi_2$ and Φ are equisatisfiable in $T_1 \cup T_2$.

The variables shared by Φ_1 and Φ_2, i.e. those in $Var(\Phi_1) \cap Var(\Phi_2)$, are called the *interface variables* in $\Phi_1 \wedge \Phi_2$. If x_i and x_j are two interface variables, then the equality $x_i = x_j$ is called an *interface equality*, and is denoted by e_{ij}. As $x_i = x_j$ and $x_j = x_i$ are semantically equivalent, we implicitly consider only one of them, e.g., the one for which $i < j$. The purification process can be easily lifted from conjunctions of $\Sigma_1 \cup \Sigma_2$-literals to $\Sigma_1 \cup \Sigma_2$-formulae. Similarly, the interface variables in a pure $\Sigma_1 \cup \Sigma_2$-formula ϕ are those that are shared between the 1- and 2-pure atoms in ϕ. In the following, we denote with $IE(\phi)$ the set of its interface equalities. In the rest of this section, without loss of generality, we consider the satisfiability of formulae of the form $\Phi_1 \wedge \Phi_2$, where Φ_i is a conjunction of i-pure literals.

The following lemma (adapted from [37]) is the basis of the correctness of the Nelson–Oppen schemas presented in Section 2.3.

Lemma 1. *If T_1 and T_2 are two signature-disjoint stably infinite theories and Φ_i is a conjunction of i-pure literals (for $i = 1, 2$), then $\Phi_1 \wedge \Phi_2$ is $T_1 \cup T_2$-satisfiable if and only if there exists an identification σ of the interface variables in $\Phi_1 \wedge \Phi_2$ such that $\Phi_1 \wedge \widehat{\sigma}$ is T_1-satisfiable and $\Phi_2 \wedge \widehat{\sigma}$ is T_2-satisfiable.*

2.3. The Nelson–Oppen combination schemas

Lemma 1 suggests the following schema, which henceforth we refer to as "non-deterministic Nelson–Oppen" [25,36,37]. Assume that we want to check the $T_1 \cup T_2$-satisfiability of a conjunction Φ of quantifier-free $\Sigma_1 \cup \Sigma_2$-literals.

(1) As a preliminary step we purify Φ into a conjunction $\Phi_1 \wedge \Phi_2$ of pure literals. Let $x_1, ..., x_n$ be all the variables in $Var(\Phi_1) \cap Var(\Phi_2)$.
(2) We guess an identification σ over $x_1, ..., x_n$ and we consider the problem of checking the T_i-satisfiability of $\Phi_i \wedge \widehat{\sigma}$ for $i = 1, 2$.
(3) If $\Phi_i \wedge \widehat{\sigma}$ is T_i-satisfiable for both $i = 1$ and $i = 2$, then we conclude the $T_1 \cup T_2$-satisfiability of Φ.
Otherwise we go back to step 2 considering another identification of variables, if any is still unconsidered.
(4) If no more identification of variables must be considered and no satisfiability has been detected at step 2, then we return the $T_1 \cup T_2$-unsatisfiability of Φ.

Example 1. Let T_1 be $\mathcal{EUF}$ and T_2 be $\mathcal{LA}(Int)$. Suppose that we want to establish the $T_1 \cup T_2$-unsatisfiability of $\Phi_1 \wedge \Phi_2$ by using Lemma 1, where

$$\Phi_1 := f(x) \neq f(w_1) \wedge f(x) \neq f(w_2) \textit{ and}$$
$$\Phi_2 := 1 \leqslant x \wedge x \leqslant 2 \wedge w_1 = 1 \wedge w_2 = 2.$$

The set V of interface variables is $\{x, w_1, w_2\}$. There are only five cases to consider (corresponding to the five distinct equivalence relations on V):

(1) If $\widehat{\sigma}$ is $x = w_1 \wedge x = w_2 \wedge w_1 = w_2$, then $\Phi_1 \wedge \widehat{\sigma}$ is T_1-unsatisfiable because $f(x) \neq f(w_2) \wedge x = w_2$ is.
(2) If $\widehat{\sigma}$ is $x \neq w_1 \wedge x = w_2 \wedge w_1 \neq w_2$, then $\Phi_1 \wedge \widehat{\sigma}$ is T_1-unsatisfiable because $f(x) \neq f(w_2) \wedge x = w_2$ is.
(3) If $\widehat{\sigma}$ is $x = w_1 \wedge x \neq w_2 \wedge w_1 \neq w_2$, then $\Phi_1 \wedge \widehat{\sigma}$ is T_1-unsatisfiable because $f(x) \neq f(w_1) \wedge x = w_1$ is.
(4) If $\widehat{\sigma}$ is $x \neq w_1 \wedge x \neq w_2 \wedge w_1 = w_2$, then $\Phi_2 \wedge \widehat{\sigma}$ is T_2-unsatisfiable because $w_1 = 1 \wedge w_2 = 2 \wedge w_1 = w_2$ is.
(5) If $\widehat{\sigma}$ is $x \neq w_1 \wedge x \neq w_2 \wedge w_1 \neq w_2$, then $\Phi_2 \wedge \widehat{\sigma}$ is T_2-unsatisfiable because $T_2 \cup \Phi_2 \models x = w_1 \vee x = w_2$ and both $x = w_1 \wedge x \neq w_1$ and $x = w_2 \wedge x \neq w_2$ are T_2-unsatisfiable.

Since for each case we have detected either the T_1- or T_2-unsatisfiability, Lemma 1 allows us to conclude that $\Phi_1 \wedge \Phi_2$ is $T_1 \cup T_2$-unsatisfiable.

The procedure obviously terminates since only finitely many different identification σ of interface variables must be considered. The correctness is an immediate consequence of Lemma 1 and the fact that purification preserves satisfiability. Oppen [25] proved that if the satisfiability problem for T_i is in NP for both $i = 1, 2$ (and this is the case for $\mathcal{E}$, $\mathcal{LA}(Rat)$, and $\mathcal{LA}(Int)$), then the satisfiability problem for $T_1 \cup T_2$ is also in NP.

The non-deterministic combination schema above can be translated in the obvious way into a *naive deterministic* procedure by case-splitting on all the ways the interface variables in the conjunction of pure literals can be equal. Then, the individual satisfiability procedures for the component theories can be used to check the satisfiability of each branch of the split.

To design a *deterministic and efficient* combination schema to exploit Lemma 1, the satisfiability procedures for T_1 and T_2 must have some properties. They must be (i) *incremental* (i.e., it must be possible to add literals to the conjunctions without restarting the procedures), (ii) *resettable* (i.e., it must be possible to remove literals from the conjunctions without restarting the procedures), and (iii) capable of detecting all (disjunctions of) interface equalities implied by the conjunctions. We call the latter feature, *e_{ij}-deduction capability*, and we say that a solver is *e_{ij}-deduction complete* iff it is always able to perform this deduction.

If we have these three capabilities, a deterministic version of the Nelson–Oppen combination simply consists of exchanging between the two procedures the equalities between variables which are implied by the conjunctions [23]. In case disjunction of equalities are derived, case-splitting is also necessary and the capabilities (i) and (ii) are very useful.

Example 2. Let us consider again the situation of Example 1 and assume that the satisfiability procedures for T_1 and T_2 have properties (i), (ii), and (iii). The Nelson–Oppen combination schema runs as follows:

(1) The literals of Φ_1 are processed one by one, T_1-satisfiability is reported, and no equality is derived.

(2) The literals of Φ_2 are processed one by one, T_2-satisfiability is reported, and the disjunction $x = w_1 \vee x = w_2$ is returned.
(3) There is a case-splitting and the two equalities $x = w_1$ and $x = w_2$ are passed to the satisfiability procedure for T_1:
   (a) $\Phi_1 \wedge x = w_1$ is T_1-unsatisfiable, since $f(x) \neq f(w_1) \wedge x = w_1$ is;
   (b) $\Phi_1 \wedge x = w_2$ is T_1-unsatisfiable, since $f(x) \neq f(w_2) \wedge x = w_2$ is.
(4) At this point, the Nelson–Oppen method returns the $T_1 \cup T_2$-unsatisfiability of $\Phi_1 \wedge \Phi_2$.

Notice that to efficiently handle the two splittings 3(a) and 3(b), the properties (i) and (ii) are crucial, whilst the property (iii) allows us to avoid guessing equivalence relations on the interface variables.

The following remark about non-convexity and case-splitting is important. It is only necessary to propagate equalities between variables, not between variables and constants. So, conjunctions such as $1 \leqslant x \wedge x \leqslant 1000$ in $\mathcal{LA}(Int)$ will not cause one thousand splits, unless each of the numbers $1, \ldots, 1000$ is constrained equal to an interface variable.

In [25], Oppen shows that the deterministic version of the Nelson–Oppen schema based on equality exchanging for theories which are convex and admit polynomial-time satisfiability procedures runs in polynomial time. When non-convex theories are considered, case-splitting on the entailed disjunctions of equalities between the interface variables is still required and the combination schema is no longer in the polynomial class. However, Oppen claims [25] that this schema is superior to the naive deterministic schema described above.

To the best of our knowledge, both the non-deterministic and the naive deterministic schemas have been proposed only for proving theoretical properties, and have never been implemented so far, whilst the deterministic Nelson–Oppen schema, in its many variants, is actually implemented in most tools. Therefore, henceforth we refer to the latter simply as “the Nelson–Oppen schema”.

3. $SMT(T)$ and $SMT(T_1 \cup T_2)$

In this section, we discuss the traditional approach to $SMT(T_1 \cup T_2)$. This is based on two main ingredients: the first is a decision procedure for the $T_1 \cup T_2$-satisfiability of conjunctions of literals in $T_1 \cup T_2$, such as the Nelson–Oppen schema described in the previous section. The second ingredient is a generic decision procedure Bool+T for $SMT(T)$, which takes into account the boolean structure of the formula: in a nutshell, Bool+T combines an enumeration of propositional assignments with a T-satisfiability check for conjunctions of literals in T. The decision procedure for $SMT(T_1 \cup T_2)$ is obtained simply by replacing the T-satisfiability check with a $T_1 \cup T_2$-satisfiability check.

The rest of this section is devoted to the description of Bool+T, the algorithm which underlies (with different variants) several systems such as CVCLite [3], DLSAT [9], DPLL(T) [17], haRVey [12], ICS [16], MathSAT [5], TSAT++ [1], Verifun [15], Zapato [6]. We first give a naive formulation, based on total assignment enumeration, and hence a more realistic representation of a state-of-the art $SMT(T)$ procedure, based on DPLL-style assignment enumeration.

3.1. Bool+T via total model enumeration

The decision procedures described in the following rely on two simple functions. The first one is a *propositional abstraction* function, i.e., a bijective function *fol2prop* which maps a ground first-order formula ϕ into a boolean formula ϕ^p, as follows: *fol2prop* maps boolean atoms into themselves, ground atoms into fresh boolean atoms, and is homomorphic w.r.t. boolean operators and set inclusion. The second function, *prop2fol*, is called *refinement* and is the inverse of *fol2prop*. (Both functions can be implemented in such a way to require constant time to map a literal from one representation to the other.) In the following, sat and unsat denote the possible values returned by a satisfiability procedure; β^p is used to denote a propositional assignment; π is used to denote a conjunction of first order literals, and π^p its boolean abstraction; in general, we use the superscript p to denote boolean formulas or assignments (e.g., given an expression e, we write e^p to denote *fol2prop*(e)). We represent propositional assignments β^p indifferently as sets of propositional literals $\{l_i\}_i$ or as conjunctions of propositional literals $\bigwedge_i l_i$; in both cases, the intended meaning is that a positive literal v (resp. a negative literal $\neg v$) denotes that the variable v is assigned to true (resp. false).

Fig. 1 presents the naive version of Bool+T. The algorithm enumerates the total truth assignments for (the propositional abstraction of) ϕ, and checks for satisfiability in T. The procedure concludes satisfiability if an assignment is T-*satisfiable*, or returns with failure otherwise. The function *pick_total_assign* returns a truth assignment β^p to all the propositional variables in $\mathcal{A}^p$ which satisfies ϕ^p, that is, it assigns a truth value to all variables in $\mathcal{A}^p$. (Notice that, in general, *pick_total_assign* assigns a truth value to all atoms occurring in the set received as first argument, no matter if such atoms occur in the formula received as second argument or not.) The function T-*satisfiable*(β) detects if the set of conjuncts β is T-satisfiable: if so, it returns (sat, $\emptyset$); otherwise, it returns (unsat, π), where *fol2prop*(π) $\subseteq \beta^p$ and π is a T-unsatisfiable set, called a *theory conflict set*. We call $\neg$*fol2prop*(π) a *conflict clause*.

```
function Bool+$T$ ($\phi$: quantifier-free formula)
1    $\phi^p \longleftarrow fol2prop(\phi)$;
2    $\mathcal{A}^p \longleftarrow fol2prop(Atoms(\phi))$;
3    while Bool-$satisfiable(\phi^p)$ do
4        $\beta^p \longleftarrow pick\_total\_assign(\mathcal{A}^p, \phi^p)$;
5        $(\rho, \pi) \longleftarrow T\text{-}satisfiable(prop2fol(\beta^p))$;
6        if ($\rho ==$ sat) then return sat;
7        $\phi^p \longleftarrow \phi^p \wedge \neg fol2prop(\pi)$;
8    end while;
9    return unsat;
end function
```

Fig. 1. A naive Bool+T procedure based on total model enumeration.

```
function Bool+$T$ ($\phi$: quantifier-free formula)
1   $\phi \longleftarrow Preprocess(\phi)$;
2   $\phi^p \longleftarrow fol2prop(\phi)$; $\mathcal{A}^p \longleftarrow fol2prop(Atoms(\phi))$;
3   $\beta^p \longleftarrow \emptyset$;
4   while (true) do
5     repeat
6       $(\rho, \pi^p) \longleftarrow$ BCP();
7       if ($\rho ==$ unsat) then                      // boolean conflict
8         backjump&learn ($\pi^p$);                    // boolean backjumping&learning
9         if (no_open_branches) then return unsat;
10      else
11        $(\rho, \pi, \gamma) \longleftarrow T$-satisfiable($prop2fol(\beta^p)$);
12        if ($\rho ==$ unsat) then                    // early pruning
13          backjump&learn ($fol2prop(\pi)$);   // theory backjumping&learning
14          if (no_open_branches) then return unsat;
15        else
16          if ($depth(\beta^p) == |\mathcal{A}^p|$) then    // satisfying total assignment
17            return sat;
18          else                                     // theory deduction;
19            deduce&learn($fol2prop(\gamma)$);
20    until (fixpoint on $\beta^p$);
21    decide_new_branch ();
22  end while;
end function
```

Fig. 2. A Bool+T procedure based on DPLL-based model enumeration.

3.2. Bool+T *via DPLL-based model enumeration*

Fig. 2 presents a concrete, DPLL-based implementation of the algorithm defined in the previous section. [1] In Bool+T, β^p is implemented as a stack of literals, which are labeled either as *open decision*, *closed decision*, or *implied.* (To simplify the notation, we implicitly assume that $\mathcal{A}^p$, ϕ^p and β^p are global variables, so that every function accesses and possibly modifies them.) Bool+T uses the following functions:

- *Preprocess* performs some preprocessing to the input formula, including CNF-ization if ϕ is not in CNF, and some simplification steps.

[1] For the sake of simplicity, in the following we will omit some technical details about DPLL tools, which can be found, e.g., in [22].

- BCP implements standard boolean Constraint Propagation of SAT solvers. The literals l's occurring as unit clauses in ϕ^p are detected and put in a queue. BCP repeatedly adds to β^p all the literals l's in the queue, assigns them to true and simplifies ϕ^p accordingly (that is, it "hides" the clauses containing positive occurrences of l, and hides the negative occurrences of l, revealing new unit clauses if any); if no conflicts are detected, then BCP returns $(\mathsf{sat}, \emptyset)$, otherwise it returns (unsat, π^p), $\pi^p \subseteq \beta^p$ being the boolean conflict set causing the conflict. Boolean conflict sets are generated by techniques based on *implication graphs* (see [33,8,22] for details).
- *backjump&learn* implements standard DPLL backjumping and learning mechanisms [33,8,22]. First (learning), starting from a conflict set $\pi^p \subseteq \beta^p$, *backjump&learn* adds the conflict clause $\neg\pi^p$ to ϕ^p. Since then, whenever all but one literal in π^p are assigned, the remaining one will automatically be assigned to false by BCP.
  Second (backjumping), *backjump&learn* pops β^p up to the most recent open decision literal s.t. at least one literal in π^p is no more in β^p; if no such literal exists, then it sets a flag "*no_open_branches*" to true; otherwise, it flips the value of the open decision literal and closes it. This prunes all open branches below that decision point in the search tree. [2]
- *T-satisfiable* differs from that of Section 3.1 by the fact that it may have *deduction capabilities*; that is, in case of T-satisfiable input *prop2fol* (β^p), it returns $(\mathsf{sat}, \pi, \gamma)$, where γ contains a set of literals l_i on unassigned atoms in *Atoms* (ϕ) which are *deduced* in the theory T from *prop2fol* (β^p), and a set of implications $\gamma_i \rightarrow l_i$ (with $\gamma_i \subset$ *prop2fol* (β^p)) which caused the deduction, i.e, s.t. $\gamma_i \models_T l_i$ for every i.
- *deduce&learn* adds to the BCP queue every literal *fol2prop*(l_i) in γ, so that its value will be assigned and propagated by the next run of BCP. Moreover, is adds *fol2prop*$(\neg\gamma_i \vee l_i)$ (*deduction clause* hereafter) to ϕ^p, obtaining an effect analogous to that of learning.
- *decide_new_branch* picks an unassigned literal l according to some heuristic criterion, declares it an open decision literal, and adds it to the BCP queue.

On the whole, $\mathsf{Bool}+T$ tries to build a total truth assignment β^p which satisfies ϕ^p s.t. *prop2fol* (β^p) is satisfiable in T. Initially, the formula ϕ is preprocessed and $\mathcal{A}^p$, ϕ^p, and β^p are initialized. Let us consider a generic point in the search tree.

- BCP is applied first, so as to assign true to the literals in the BCP queue (either unit clauses or literals enqueued by some other function).
- If a boolean conflict π^p is generated, then $\mathsf{Bool}+T$ backjumps and learns the given conflict clause (we call this step *boolean backjumping and learning*). If no open branches are left, which means that all candidate assignments have been investigated, then $\mathsf{Bool}+T$ returns unsat. Otherwise, the satisfiability in T of (the refinement of) the current assignment β^p is tested by *T-satisfiable*.
- If *T-satisfiable* returns unsat, then no total assignment extending β^p can be refined into a consistent set of literals, and $\mathsf{Bool}+T$ backtracks. (We call this step *early pruning*.) The theory conflict

[2] Notice that this is a simplified description: modern DPLL tools may flip the value of non-decision literals as well (see [22] for details).

set generated causes Bool+T to backjump and learn the given conflict clause, and possibly to stop, as in steps 7–9. We call this step *theory-driven backjumping and learning.*
- If instead *T-satisfiable* returns sat, then if β^p is a total assignment, then Bool+T returns sat;[3] if not, *deduce&learn* is invoked on the literals deduced by *T-satisfiable* and on their relative reasons in γ, producing new unit literals for BCP and new deduction clauses. We call this step *theory-driven deduction.*

Steps from 6 to 19 are repeated until this causes no more modifications of β^p. When this happens, a new literal is selected by *decide_new_branch*, and the procedure restarts from step 5.

The idea underlying Bool+T is to use a DPLL solver as an optimized truth assignment enumerator. We notice that the procedure in Fig. 2 enhances that in Fig. 1 by cutting *partial* assignments which cannot be expanded into satisfying total ones. This is done in two ways. First, it cuts partial assignments which *propositionally falsify* the input formula. This is done by means of BCP, boolean backjumping and learning, and it is well-known from the SAT literature. Second, it cuts partial assignments whose refinement *are inconsistent in the theory T*. This is done by means of early pruning, theory-driven backjumping and learning, and theory-driven deduction. *Early pruning* allows for cutting every partial assignment as soon as (its refinement) becomes inconsistent in T, thus pruning the whole set of total assignments which extend it. *Theory-driven backjumping and learning* allows for cutting every partial assignment containing a previously detected theory conflict set π^p, thus pruning all but one total assignments containing π^p. *Theory-driven deduction* allows for cutting every partial assignments which assigns false to one deduced literal l, thus pruning the whole set of total assignments which do not contain l.

The improvements caused by learning the deduction clauses are manifold, depending on the theory addressed. First, in some theories theory-deduction can be very expensive, so that learning the deduction clauses allows for performing each deduction only once, leaving the others to the (much faster) BCP. Second, in some theories (e.g., $\mathcal{EUF}$ [24]), the solvers can efficiently perform deductions of equalities, like $\{x = y, y = z\} \models f(x) = f(z)$, but not of disequalities, like $\{x = y, \neg(f(x) = f(z))\} \models \neg(y = z)$. In the first case, learning the deduction clause $\neg(x = y) \vee \neg(y = z) \vee (f(x) = f(z))$ will allow BCP also to propagate $\neg(y = z)$ from $\{x = y, \neg(f(x) = f(z))\}$. Third, in non-convex theories, some solvers can perform deductions of *disjunctions* of equalities $\bigvee x_i = x_j$ from sets of literals $\{l_1, ..., l_k\}$. Learning the deduction clause $\neg l_1 \vee \cdots \vee \neg l_k \vee \bigvee x_i = x_j$ allows for using also these deductions to prune the boolean search. Finally, an improvement w.r.t. theory-driven learning is that the deduction clause may be learned earlier, when all but one literals in the theory conflict set have been assigned, without waiting the SAT solver to assign the last literal.

Example 3. Let T_1 be $\mathcal{EUF}$ and T_2 be $\mathcal{LA}(Int)$, and consider the following *SMT* problem for the pure formula Φ:

$$\neg(f(x) = f(w_1)) \wedge (A \leftrightarrow \neg(f(x) = f(w_2))) \wedge 1 \leqslant x \wedge x \leqslant 2 \wedge w_1 = 1 \wedge w_2 = 2$$

[3] Notice that here the totality of β^p is not strictly necessary, as it would be sufficient to test if the current interpretation satisfies propositionally the formula. However, the test for totality is somewhat standard in state-of-the-art DPLL solvers, due to efficiency reasons.

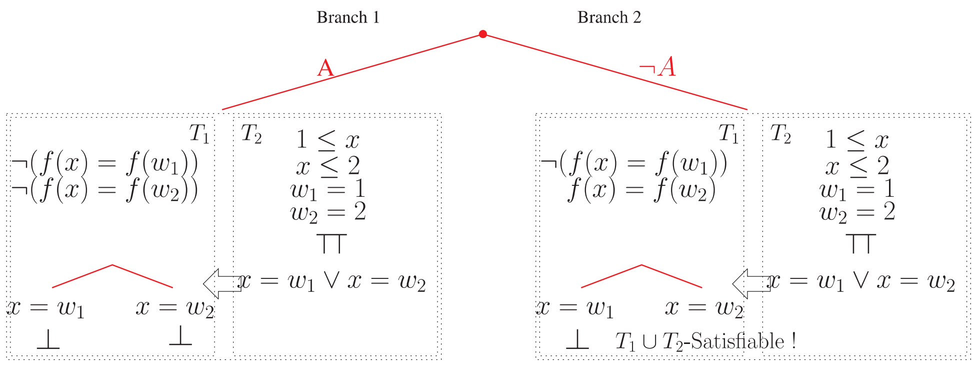

Fig. 3. Representation of the search tree for Bool+no(T_1, T_2) applied to $\Phi = \neg(f(x) = f(w_1)) \wedge (A \leftrightarrow \neg(f(x) = f(w_2))) \wedge 1 \leqslant x \wedge x \leqslant 2 \wedge w_1 = 1 \wedge w_2 = 2$.

$V = \{x, w_1, w_2\}$ being the set of interface variables. Fig. 3 represents the search tree for Bool+no(T_1, T_2) applied to Φ.

Suppose we first assign the boolean variable A to true (branch 1), so that Φ simplifies into the formula $\Phi_1 \wedge \Phi_2$ of Examples 1 and 2. The no(T_1, T_2) schema runs as in Example 2, finding a contradiction.

Then we assign A to false (branch 2), so that Φ simplifies into:

$$\overbrace{\neg(f(x) = f(w_1)) \wedge f(x) = f(w_2)}^{\Phi_1'} \wedge \overbrace{1 \leqslant x \wedge x \leqslant 2 \wedge w_1 = 1 \wedge w_2 = 2}^{\Phi_2}.$$

Φ_1' differs from Φ_1 as $f(x) = f(w_2)$ is not negated. Hence the no(T_1, T_2) combination schema reruns steps 1, 2, and 3(a) of Example 2, like in branch 1. Then, in step 3(b), $x = w_2$ is passed to the satisfiability procedure for T_1, which states that $\Phi_1' \wedge x = w_2$ is T_1-satisfiable, and no more interface equality can be deduced. Thus $\Phi_1' \wedge \Phi_2$ is $T_1 \cup T_2$-satisfiable, and hence Φ is $T_1 \cup T_2$-satisfiable.

In general, SMT tools can learn only clauses containing only atoms occurring in the input formula.[4] Thus, e_{ij}'s not occurring in the original formula cannot occur in learned clauses. Therefore, generating conflict clauses in $T_1 \cup T_2$ within the no(T_1, T_2) schema is a complex process, involving a backward chain of resolution steps on both the conflict clauses found and the deductions performed in the single theories, so that to eliminate interface equalities from them. Moreover, such conflict clauses may be very long and are of no help for the deduction of interface equalities.

In Example 3, the minimal clause not containing interface equalities which can be obtained in branch 1 is[5] $\neg(1 \leqslant x) \vee \neg(x \leqslant 2) \vee \neg(w_1 = 1) \vee \neg(w_2 = 2) \vee (f(x) = f(w_1)) \vee (f(x) = f(w_2))$, that is, $\neg(\Phi_1 \wedge \Phi_2)$ itself, which is of no help for pruning the search in branch 2.

[4] A noteworthy exception to this fact is VERIFUN [15].

[5] This is the minimal conflict clause obtained by resolving the deduction of step (2) in Example 2, $\{1 \leqslant x, x \leqslant 2, w_1 = 1, w_2 = 2\} \models (x = w_1 \vee x = w_2)$, with the conflict clauses obtained by steps 3 (a) and 3 (b), $\neg(x = w_1) \vee f(x) = f(w_1)$ and $\neg(x = w_2) \vee f(x) = f(w_2)$.

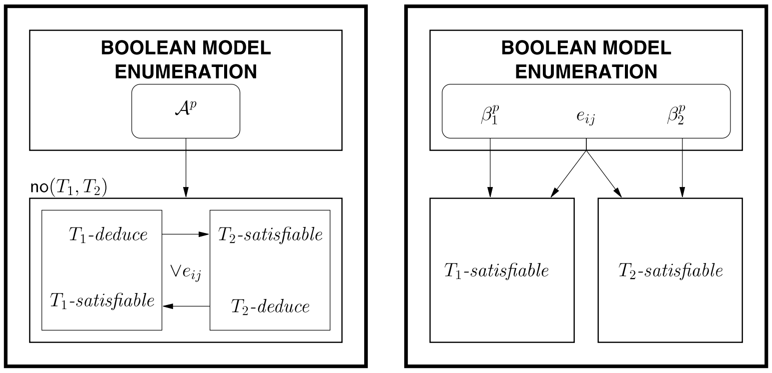

Fig. 4. The different schemas for $SMT(T_1 \cup T_2)$.

4. $SMT(T_1 \cup T_2)$ via Delayed Theory Combination

We propose a new approach to the $SMT(T_1 \cup T_2)$ problem by taking a different view: we get rid of the Nelson–Oppen schema (see Fig. 4, left), tighten the coupling between the boolean solver Bool and each satisfiability procedure T_i*-satisfiable*, and delay the combination of the theories (see Fig. 4, right). The new approach does not require the direct combination of decision procedures for T_1 and T_2. The boolean solver Bool is coupled with a satisfiability procedure T_i*-satisfiable* for each T_i, and each of the theory 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 of the form e_{ij}. Both theory solvers receive, from the boolean level, the same truth assignment 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. We call the approach *Delayed Theory Combination* (DTC), since we can delay the "synchronization" between the satisfiability checks in the component theories by lifting the phase of deducing equalities to the propositional level, by introducing "fresh" propositional atoms encoding these special equalities.

4.1. Bool+T_1+T_2 *via total model enumeration*

We use notations similar to those used in Section 3. In addition, we use a subscript i to refer to the theory T_i; each assignment β^p is split into $\beta_1^p \wedge \beta_2^p \wedge \beta_e^p$, where *prop2fol* (β_i^p) is a set of i-pure literals and *prop2fol* (β_e^p) is a set of e_{ij}-literals. The functions Bool-*satisfiable*, *pick_total_assign*, T_i*-satisfiable*, BCP, *backjump&learn*, *deduce&learn*, *decide_new_branch* are analogous to those in Section 3. The function *purify*(ϕ) returns a formula ϕ' containing only i-pure literals which is equisatisfiable to ϕ. (We call ϕ' the *purified* version of ϕ.) The function $IE(\phi)$ constructs the set $\{x_1, ..., x_n\}$ of interface variables in ϕ, and then generates the $n(n-1)/2$ interface equalities $x_i = x_j$, $1 \leqslant i < j \leqslant n$.

Fig. 5 represents a naive version of Bool+T_1+T_2, a decision procedure for $SMT(T_1 \cup T_2)$. Initially (lines 1 and 2), the formula is purified, the interface variables x_i are identified and the interface equal-

```
function Bool+$T_1$+$T_2$ ($\phi$: quantifier-free formula)
1    $\phi \longleftarrow purify(\phi)$;
2    $\phi^p \longleftarrow fol2prop(\phi)$; $\mathcal{A}^p \longleftarrow fol2prop(Atoms(\phi) \cup IE(\phi))$;
3    while Bool-satisfiable ($\phi^p$) do
4        $\beta_1^p \wedge \beta_2^p \wedge \beta_e^p = \beta^p \longleftarrow pick\_total\_assign(\mathcal{A}^p, \phi^p)$;
5        $(\rho_1, \pi_1) \longleftarrow T_1$-satisfiable $(prop2fol(\beta_1^p \wedge \beta_e^p))$;
6        $(\rho_2, \pi_2) \longleftarrow T_2$-satisfiable $(prop2fol(\beta_2^p \wedge \beta_e^p))$;
7        if ($\rho_1 ==$ sat and $\rho_2 ==$ sat) then return sat;
8        if ($\rho_1 ==$ unsat) then $\phi^p \longleftarrow \phi^p \wedge \neg fol2prop(\pi_1)$;
9        if ($\rho_2 ==$ unsat) then $\phi^p \longleftarrow \phi^p \wedge \neg fol2prop(\pi_2)$;
10   end while;
11   return unsat;
end function
```

Fig. 5. A naive enumeration-based $T_1 \cup T_2$-satisfiability procedure: Bool+T_1+T_2.

ities e_{ij} are created by $IE(\phi)$, and added to the set of propositional symbols $\mathcal{A}^p$, and the propositional abstraction ϕ^p of ϕ is created.

The main loop (lines 3–10) enumerates propositional assignments on the extended atoms set $\mathcal{A}^p$, which are then checked for both T_1- and T_2-consistency. While ϕ^p is propositionally satisfiable (line 3), we select a truth assignment β^p over $\mathcal{A}^p$ that satisfies ϕ^p, and we split it into $\beta_1^p \wedge \beta_2^p \wedge \beta_e^p$ (line 4). Notice that β^p assigns also (the boolean abstraction of) e_{ij} literals which do not occur in ϕ. For each theory T_i, *prop2fol* $(\beta_i^p \wedge \beta_e^p)$ (i.e., the part of β^p which is relevant for T_i) is checked for T_i-consistency (lines 5 and 6). If both calls to 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. $\pi_i \subseteq \beta$ and π_i is T_i-unsatisfiable. Then, as in the case of Bool+T, ϕ^p is strengthened to exclude truth assignments which may fail for the same reason (lines 8 and 9), and the loop is resumed. Unsatisfiability is returned (line 11) when the loop is exited without having found a model.

The procedure in Fig. 5 differs from that of Fig. 1 for the following facts. First, the formula ϕ is purified (line 1). Second, the set of atoms $\mathcal{A}^p$ to assign is extended to also include all interface equalities e_{ij}, no matter whether they explicitly occur in the purified formula ϕ or not (line 2); thus, the new procedure looks for assignments β^p's which give a truth value to all interface equalities as well. Third, (the refinement of) one such total assignment β^p is decided to be $T_1 \cup T_2$-consistent iff *prop2fol* $(\beta_i^p \wedge \beta_e^p)$ is T_i-consistent, for both i's (lines 5–9).

4.2. Bool+T_1+T_2 *via DPLL-based enumeration*

Fig. 6 presents one concrete DPLL-based representation of the Bool+T_1+T_2 algorithm. Here the enumeration of assignments is carried out by means of a DPLL-based SAT engine, and all the optimizations discussed for Bool+T can be retained. The input formula ϕ is preprocessed (line 1); then, as in Fig. 5, ϕ is purified, the interface equalities e_{ij} are created by $IE(\phi)$, and added to the set of propositional symbols $\mathcal{A}^p$, and ϕ^p and β^p are initialized.

```
function Bool + $T_1$ + $T_2$ ($\phi$: quantifier-free formula)
1   $\phi \longleftarrow Preprocess(\phi)$;
2   $\phi \longleftarrow purify(\phi)$;
3   $\phi^p \longleftarrow fol2prop(\phi)$; $\mathcal{A}^p \longleftarrow fol2prop(Atoms(\phi) \cup IE(\phi))$;
4   $\beta^p \longleftarrow \emptyset$;
5   while (true) do
6     repeat
7       $(\rho, \pi^p) \longleftarrow$ BCP();
8       if ($\rho$ == unsat) then                          // boolean conflict
9         backjump&learn ($\pi^p$);                       // boolean backj.&learning
10        if (no_open_branches) then return unsat;
11      else
12        $\beta_1^p \wedge \beta_e^p \wedge \beta_2^p = \beta^p$;
13        $(\rho_1, \pi_1, \gamma_1) \longleftarrow T_1$-satisfiable($prop2fol(\beta_1^p \wedge \beta_e^p)$);
14        if ($\rho_1$ == unsat) then                     // early pruning
15          backjump&learn ($fol2prop(\pi_1)$);   // theory backj.&learning
16          if (no_open_branches) then return unsat;
17        else
18          $(\rho_2, \pi_2, \gamma_2) \longleftarrow T_2$-satisfiable($prop2fol(\beta_e^p \wedge \beta_2^p)$);
19          if ($\rho_2$ == unsat) then                   // early pruning
20            backjump&learn ($fol2prop(\pi_2)$);   // theory backj.&learning
21            if (no_open_branches) then return unsat;
22          else
23            if ($depth(\beta^p) == |\mathcal{A}^p|$) then      // satisfying total assignment
24              return sat;
25            else                                        // theory deduction;
26              deduce&learn($fol2prop(\gamma_1)$);
27              deduce&learn($fol2prop(\gamma_2)$);
28    until (fixpoint on $\beta^p$);
29    decide_new_branch ();
30  end while
end function
```

Fig. 6. One DPLL-based representation of the Delayed Theory Combination Bool+T_1+T_2.

The remaining part of the procedure is analogous to that in Fig. 2, modified to deal with the fact that a candidate total assignment $\beta^p = \beta_1^p \wedge \beta_e^p \wedge \beta_2^p$ over the extended set $\mathcal{A}^p$ is $T_1 \cup T_2$-consistent iff *prop2fol* $(\beta_i^p \wedge \beta_e^p)$ is T_i-consistent, for $i = 1, 2$, so that any candidate partial assignment can be dropped as soon as it no longer matches one such condition. Thus:

- The global structure of two loops (lines 4, 5, 20–22 in Fig. 2) is maintained in Fig. 6 (lines 5, 6, 28–30), as well as the nested if-then-else structure of the internal loop.
- BCP, the detection of boolean conflicts and the boolean backjumping and learning (lines 6–9 in Fig. 2) are kept identical (lines 7–10 in Fig. 6).
- The $\mathcal{T}$*-satisfiable* test, early pruning, theory backjumping&learning and possible failure (lines 11–14 in Fig. 2) are duplicated in Fig. 6 for both theories $\mathcal{T}_1$ (lines 13–16) and $\mathcal{T}_2$ (lines 18–21).
- The success condition (lines 16 and 17 in Fig. 2) are kept identical (lines 23 and 24 in Fig. 6).
- The theory deduction step (lines 18 and 19 in Fig. 2) is mapped into the concatenation of the deduction steps for the two distinct theories (lines 25–27 in Fig. 6).

It is important to notice that both the conflict clauses and the deduction clauses which are learned during the search may contain interface equalities e_{ij}'s.

For the sake of efficiency, we assume that all theory solvers are *incremental* and *resettable*, as with N.O. (properties (i) and (ii) in Section 2.3).[6] However, unlike with N.O., we do not require that they are also e_{ij}-deduction complete (property (iii)), although DTC benefits from e_{ij}-deduction capabilities.

Example 4. Consider the *SMT* problem of Example 3. Fig. 7 (top) represents the search tree of Bool+$\mathcal{T}_1$+$\mathcal{T}_2$ applied to Φ, under the hypothesis that the theory solvers have no e_{ij}-deduction capability. We suppose that the SAT solver branches, in order, on A, $(w_1 = w_2)$, $(x = w_1)$, $(x = w_2)$, assigning them the true value first. We first assign A to true (branch 1), so that $(f(x) = f(w_2))$ is false.

(1) Choosing $(w_1 = w_2)$ causes a $\mathcal{T}_2$-inconsistency be revealed by the early-pruning call to $\mathcal{T}_2$*-satisfiable*; the clause C_1: $\neg(w_1 = 1) \vee \neg(w_2 = 2) \vee \neg(w_1 = w_2)$ is learned, and Bool+$\mathcal{T}_1$+$\mathcal{T}_2$ backtracks to $\neg(w_1 = w_2)$, which does not cause inconsistency.
(2) Similarly, choosing $(x = w_1)$ causes a $\mathcal{T}_1$-inconsistency, the conflict clause C_2: $\neg(x = w_1) \vee f(x) = f(w_1)$ is learned, and Bool+$\mathcal{T}_1$+$\mathcal{T}_2$ backtracks to $\neg(x = w_1)$, which does not cause inconsistency.
(3) Similarly, choosing $(x = w_2)$ causes a $\mathcal{T}_1$-inconsistency, the conflict clause C_3: $\neg(x = w_2) \vee f(x) = f(w_2)$ is learned, and Bool+$\mathcal{T}_1$+$\mathcal{T}_2$ backtracks to $\neg(x = w_2)$.
(4) $\neg(x = w_2)$ causes a $\mathcal{T}_2$-inconsistency, so that branch 1 is closed, and the clause C_4: $\neg(1 \leqslant x) \vee \neg(x \leqslant 2) \vee \neg(w_1 = 1) \vee \neg(w_2 = 2\} \vee x = w_1 \vee x = w_2$ is learned.

Then we assign A to false (branch 2), so that $f(x) = f(w_2)$ is true. Because of C_1, C_2 and C_4 respectively, $\neg(w_1 = w_2)$, $\neg(x = w_1)$ and $(x = w_2)$ are immediately assigned by BCP without causing inconsistencies, so that Φ is $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable.

Fig. 7 (middle) represents the search tree of Bool+$\mathcal{T}_1$+$\mathcal{T}_2$ applied to Φ, under the hypothesis that the theory solvers are e_{ij}-deduction complete. $\mathcal{T}_2$*-satisfiable* performs the deduction $\{1 \leqslant x, x \leqslant 2, w_1 = 1, w_2 = 2\} \models (x = w_1 \vee x = w_2)$ and the corresponding deduction clause is learned, which is identical to C_4 above. As before, we first assign A to true (branch 1), so that $(f(x) = f(w_2))$ is false. As before, we suppose that the SAT solver branches, in order, on $(w_1 = w_2)$, $(x = w_1)$, $(x = w_2)$, assigning them the true value first. If so, the search tree proceeds as in the previous case until

[6] These are highly desirable features in SMT tools in general [5].

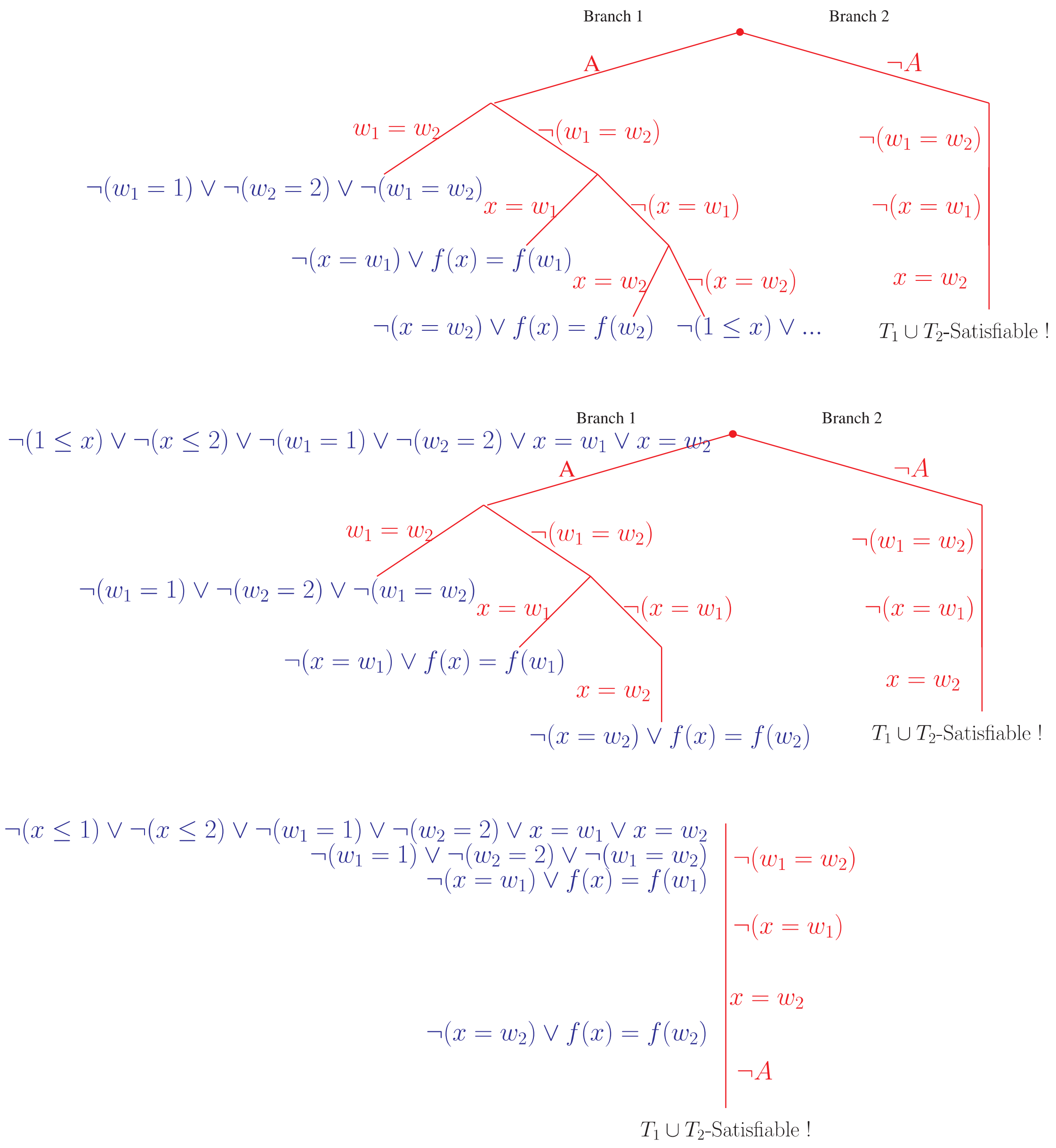
Branch 1
Branch 2
A
¬A
w1 = w2
¬(w1 = w2)
¬(w1 = 1) ∨ ¬(w2 = 2) ∨ ¬(w1 = w2)
x = w1
¬(x = w1)
¬(x = w1) ∨ f(x) = f(w1)
x = w2
¬(x = w2)
¬(x = w2) ∨ f(x) = f(w2)
¬(1 ≤ x) ∨ ...
T1 ∪ T2-Satisfiable !
¬(1 ≤ x) ∨ ¬(x ≤ 2) ∨ ¬(w1 = 1) ∨ ¬(w2 = 2) ∨ x = w1 ∨ x = w2
¬(x ≤ 1) ∨ ¬(x ≤ 2) ∨ ¬(w1 = 1) ∨ ¬(w2 = 2) ∨ x = w1 ∨ x = w2

Fig. 7. Representation of the search tree for Bool+T_1+T_2 applied to $\Phi = \neg(f(x) = f(w_1)) \wedge (A \leftrightarrow \neg(f(x) = f(w_2))) \wedge 1 \leqslant x \wedge x \leqslant 2 \wedge w_1 = 1 \wedge w_2 = 2$. Top: without deduction. Middle: with e_{ij}-deduction. Bottom: with complete deduction.

$\neg(w_1 = w_2)$, $\neg(x = w_2)$ are assigned. Then $(x = w_2)$ is assigned by BCP because of the deduction clause learned C_4, and hence the search proceeds as in the previous case.

Fig. 7 (bottom) represents the search tree of Bool+T_1+T_2 applied to Φ, under the hypothesis that both theory solvers are theory-deduction complete (e.g., they can deduce negative equalities as well). Before splitting, T_2*-satisfiable* performs not only the deduction $\{1 \leqslant x, x \leqslant 2, w_1 = 1, w_2 = 2\} \models (x = w_1 \vee x = w_2)$, as before, but also $\{w_1 = 1, w_2 = 2\} \models \neg(w_1 = w_2)$, learning C_4 and C_1 as deduction clauses; T_1*-satisfiable* performs the deduction $\{\neg(f(x) = f(w_1))\} \models \neg(x = w_1)$, learning C_2. This causes the assignment of $\neg(w_1 = w_2)$, $\neg(x = w_1)$ and $x = w_2$ by BCP. Then T_1*-satisfiable* performs the deduction $\{x = w_2\} \models f(x) = f(w_2)$, from which $f(x) = f(w_2)$ and $\neg A$ are assigned by BCP. The resulting assignment is thus $T_1 \cup T_2$-satisfiable.

From Examples 3 and 4 we notice a few facts.

First, in DTC, using a DPLL-like engine, together with techniques like early pruning and theory-driven learning and deduction, allows for a drastic pruning of the boolean search on e_{ij}'s with respect to the enumeration of total assignments (16 in Example 4). In particular, the fact that learned clauses may contain e_{ij}'s is essential, because it makes learned clauses shorter—and thus more effective—and because it allows for reusing the information on e_{ij}'s which is learned in one branch to prune the search on e_{ij}'s in the subsequent branches.

Second, in DTC the extra boolean component of search caused by the non-convexity of $\mathcal{LA}(Int)$ is merged into the top-level boolean search, so that it is handled efficiently by the top-level DPLL procedure.

Third, unlike with N.O., with DTC we do not require e_{ij}-deduction capabilities from the theory solvers, although e_{ij}-deduction helps cutting the boolean search space by foreseeing theory conflicts. Thus, on one extreme (no e_{ij}-deductive power for theory solvers), DTC works as a SMT implementation of non-deterministic N.O. schema, with the DPLL solver playing the double role of (very efficient) truth assignment enumerator for Φ and of "smart guesser" of identifications on interface variables. On the other extreme (both theory solvers are e_{ij}-deduction complete), it behaves similarly to an SMT tool with a deterministic N.O. engine, with the added benefits we have mentioned above; every intermediate situation between the two extremes is possible, by trading DPLL boolean search for theory-deduction at will.

More generally, with DTC we impose no restriction on the theory-deduction capability of the theory solvers used, although we fully benefit from any such capability. Thus, one is free to choose and implement the procedures for T-satisfiability for the theories of interest, evaluating the tradeoff between the search-pruning benefit and the cost of theory-deduction. [7]

4.3. Discussion

In the following, we discuss the advantages of DTC.

Simplicity. The overall schema is extremely simple. Nothing is needed beyond decision procedures for each T_i, and no complicated integration schema between the T_i is required. Furthermore, when the input problem is fully contained within one T_i, the setup reduces nicely to *SMT* (T_i). All features from the DPLL framework such as early pruning, theory driven backjumping and learning, deduction, and split control can be used.

[7] E.g., with some theories such as $\mathcal{EUF}$, theory-deduction of equalities can be computationally cheap, whilst with some other theories such as $\mathcal{LA}(Rat)$, theory-deduction of equalities requires methods which are more expensive than those for simple satisfiability.

Bool vs. theory. The interaction between the boolean level and each theory is tightened, thus taking into account the fact that the boolean structure of the quantifier-free formula can severely dominate the complexity of $T_1 \cup T_2$-satisfiability. In contrast, Nelson–Oppen privileges the link between T_1 and T_2, while $SMT(T_1 \cup T_2)$ problems may feature complex interactions between the boolean level and each of the T_i. In fact, the boolean component is far from negligible with respect to the integrated theories.

Non-convexity. The DTC schema captures in a very natural way the case of non-convex theories. The Nelson–Oppen schema implements case-splitting on the disjunction of interface equalities entailed by each T_i and this case splitting is separate from the management of the boolean splitting. Therefore, the combination schema becomes very complex: one has to deal with the fact that disjunctions of e_{ij} need to be exchanged. Besides complicating the deduction mechanism of each theory, a stack-based search with backtracking has to be performed.

In DTC the search on the "top-level" boolean component of the problem and the search on the "non-convex" component are dealt with in an "amalgamated" framework, and positively interact with each other, so that to maximize the benefit of the optimizations of state-of-the art SAT procedures.

Generation of theory conflict. The construction of conflict sets may be a non trivial task within a single theory. The problem is even harder in the case of $T_1 \cup T_2$, since the construction of a conflict set must take into account the conflicts obtained in each theory, as well as the interface equalities that have been exchanged. In our framework, this complication is avoided altogether: a conflict for the combined theories is naturally induced by the interaction between the conflict in one theory and the mechanisms for conflict management in the boolean search. Moreover, DTC allows for learning clauses containing e_{ij}'s in a very natural way, which provides big benefits in further pruning the search.

Deduction. The N.O. schema requires the theory solvers being e_{ij}-deduction complete. However, e_{ij}-deduction completeness can be sometimes hard to achieve (e.g., it may greatly complicate the satisfiability algorithms), and computationally expensive to carry out. In the DTC approach, the theory solvers do not have to be deduction-complete, although DTC benefits from any e_{ij}-deduction capability. This enables us to explore the trade-off between which deduction is beneficial to efficiency and which is in fact hampering the search—or too difficult to implement.

As possible drawbacks, we notice that in DTC the *whole* formula is purified, with the corresponding upfront introduction of $O(n^2)$ interface equalities e_{ij}. This may increase the boolean search space, and increase the number of theory literals given as input to each theory solver. However, many of these may not occur in the purified formula; and even though the truth assignment of the interface equalities has to be guessed by the boolean level, which potentially increases the boolean search space, early pruning, learning and deduction help to limit the increase in the search.

4.4. Formal properties

In this section we prove that the Bool+T_1+T_2 algorithm is a decision procedure for $SMT(T_1 \cup T_2)$. We first show that the existence of a total assignment over $Atoms(\phi) \cup IE(\phi)$ is a necessary and sufficient condition for $T_1 \cup T_2$-satisfiability; then we show that the algorithm returns sat iff such an assignments exists, and returns unsat otherwise.

We state a preliminary fact to take care of formulae of generic boolean structure.

Lemma 2. *Let T be a Σ-theory, ϕ be a quantifier-free Σ-formula, and $\phi^p = fol2prop(\phi)$. Then ϕ is T-satisfiable if and only if there exists a total truth assignment β^p over $fol2prop(Atoms(\phi^p))$ such that*

($\bullet$) *β^p satisfies ϕ^p, and*
($\bullet$) *$prop2fol(\beta^p)$ is T-satisfiable.*

The lemma follows from basic notions of first-order logic—we omit the proof.

Theorem 1. *Let ϕ be a pure formula on $T_1 \cup T_2$, and let ϕ^p be $fol2prop(\phi)$.*
ϕ is $T_1 \cup T_2$-satisfiable iff there exists a total truth assignment β^p over $fol2prop(Atoms(\phi) \cup IE(\phi))$ such that

- *β^p satisfies ϕ^p,*
- *$prop2fol(\beta_1^p \wedge \beta_e^p)$ is T_1-satisfiable, and*
- *$prop2fol(\beta_2^p \wedge \beta_e^p)$ is T_2-satisfiable,*

where $\beta^p = \beta_1^p \wedge \beta_2^p \wedge \beta_e^p, \beta_1 = prop2fol(\beta_1^p)$ is T_1-pure, $\beta_2 = prop2fol(\beta_2^p)$ is T_2-pure, and $\beta_e = prop2fol(\beta_e^p)$ is a set of equality literals over $IE(\phi)$.

Proof 1.
($\Longrightarrow$) ϕ is $T_1 \cup T_2$-satisfiable. From Lemma 2, there exist a total truth assignment β^p over $Atoms(\phi^p)$ which satisfies ϕ^p, and such that $\beta = prop2fol(\beta^p)$ is $T_1 \cup T_2$-satisfiable.
Since ϕ is a $T_1 \cup T_2$-pure formula, we can write β as $\beta_1 \wedge \beta_2$, where each β_i is i-pure.
Since $\beta_1 \wedge \beta_2$ is $T_1 \cup T_2$-satisfiable, by Lemma 1 there exists a β_e over $IE(\beta_1 \wedge \beta_2)$ s.t. $\beta_i \wedge \beta_e$ is T_i-satisfiable for $i = 1, 2$.
We conclude by noticing that $IE(\beta_1 \wedge \beta_2) = IE(\phi)$.
($\Longleftarrow$) Let β^p satisfy the conditions above, and let $\beta = prop2fol(\beta^p) = \beta_1 \wedge \beta_2 \wedge \beta_e$.
By applying Lemma 1 with $\Phi_1 = \beta_1 \wedge \beta_e$ and $\Phi_2 = \beta_2 \wedge \beta_e$, we have that β is $T_1 \cup T_2$-satisfiable.
Let $\overline{\beta}$ be the restriction of β to the atoms occurring in ϕ.
Then, $\overline{\beta}$ is also $T_1 \cup T_2$-satisfiable since it is weaker than β. Furthermore, $\beta^p = prop2fol\,\overline{\beta}$ also satisfies ϕ^p, since it agrees with β^p on the atoms occurring in ϕ^p.
The thesis follows from Lemma 2. □

We now show that Bool+T_1+T_2 of Fig. 5 is a decision procedure for $SMT(T_1 \cup T_2)$. We preliminarily prove the following lemma, which states the correctness of the pruning.

Lemma 3. *Let T be a Σ-theory, ϕ be a quantifier-free Σ-formula, and $\phi^p = fol2prop(\phi)$. If β^p is a propositional assignment for ϕ^p and $\beta = prop2fol(\beta^p)$ is T-unsatisfiable, then ϕ is T-satisfiable iff $\phi \wedge \neg\beta$ is.*

Proof 2. ($\Rightarrow$) Assume that ϕ is T-satisfiable. Then there must exist an interpretation I which satisfies T and ϕ. Notice that if β is not T-satisfiable, then $T \models \neg\beta$. So, any interpretation satisfying T must also satisfy $\neg\beta$. This must hold also for I. As a consequence, we have that I satisfies T, ϕ, and $\neg\beta$. This allows us to conclude that $\phi \wedge \neg\beta$ is T-satisfiable.
($\Leftarrow$) If $\phi \wedge \neg\beta$ is T-satisfiable, then both ϕ and $\neg\beta$ are T-satisfiable. So, ϕ is T-satisfiable. □

Theorem 2. *The function* Bool+T_1+T_2 *of Fig. 5 is a satisfiability procedure for the problem of checking the* $T_1 \cup T_2$*-satisfiability of quantifier-free first-order formulae.*

Proof 3. Termination is proved by showing that in Bool+T_1+T_2 the number of iterations in the while loop (lines 3–10) is bounded by the number of total truth assignments on $\mathcal{A}^p$, which is finite. In fact, at any iteration, either the loop is exited and sat is returned (line 7), or one of step 8 or 9 are executed. Each of them strengthens ϕ^p with $\neg fol2prop(\pi_i)$ by removing (at least) the total assignment β^p currently being analyzed from the possible values returned by *pick_assignment*.

The procedure can return sat only at line 7; then, by Theorem 1 we can conclude that ϕ is $T_1 \cup T_2$-satisfiable.

The procedure can return unsat only if the test of step 7 returns false for all iterations of the loop. By lemma 3, we are guaranteed that the formula ϕ before line 7 and the formula after line 9 are equisatisfiable.
From Theorem 1 we can conclude that ϕ is $T_1 \cup T_2$-unsatisfiable by considering that we never discard a satisfying assignment for ϕ.

Finally, we notice that the Bool+T_1+T_2 procedure in Fig. 6 is also a decision procedure for *SMT*$(T_1 \cup T_2)$. Similarly to the *SMT*(T) case described in Section 3, the Bool+T_1+T_2 procedure in Fig. 6 enhances that of Fig. 5 by cutting *partial* assignments. This preserves termination, correctness and completeness. In fact, each partial assignment cannot be expanded into total ones which can cause $T_1 \cup T_2$-satisfiability. In particular, the partial assignments β^p which have been pruned by means of BCP, boolean backjumping and learning are such that they do not satisfy ϕ^p, so that no total assignment extending them would. Furthermore, the other partial assignments $\beta^p = \beta_1^p \wedge \beta_p^2 \wedge \beta_p^e$ have been pruned by means of theory-driven backjumping, learning and deduction, so that they are such that $\beta_i \wedge \beta_e$ is T_i-unsatisfiable for some i. Thus all total assignments extending them are T_i-unsatisfiable for some i, and thus they are $T_1 \cup T_2$-unsatisfiable by Theorem 1. □

5. Delayed Theory Combination in practice: MathSAT (EUF, LA)

In this section, we discuss the implementation in MathSAT [5] of the Delayed Theory Combination schema for the theories of $\mathcal{EUF}$ and $\mathcal{LA}$, both for the convex case of reals and the non-convex case of integers.

5.1. The basic platform: the MathSAT *solver*

Our starting point is MathSAT [5], an *SMT* solver able to deal with each of the following theories: $\mathcal{LA}(Rat)$, $\mathcal{LA}(Int)$, and $\mathcal{EUF}$. MathSAT is also able to deal with $(\mathcal{EUF} \cup \mathcal{LA}(Rat))$ and $(\mathcal{EUF} \cup \mathcal{LA}(Int))$ by eliminating uninterpreted symbols by means of Ackermann expansion [2].

MathSAT implements an enhanced version of the Bool+T schema (see Fig. 8, left). The SAT solver is used to enumerate models of the propositional abstraction of the formula. It interacts with two theory solvers, one for dealing with a pure $\mathcal{EUF}$-fragment and one for dealing with a pure $\mathcal{LA}$-fragment; the Ackermann expansion has made sure no interaction is needed between the theory solvers themselves. The theory solvers have a uniform interface that provides for conflict set management and deduction, and integrates backjumping and learning. The interface is also incremental and resettable; this enables the development of solvers that are state-based, and can mimic the behaviour of the SAT search. The main components of MathSAT are the following.

Preprocessor

MathSAT supports a rich input language, with a large variety of boolean and arithmetic operators, including ternary *if-then-else* constructs on the term and formula level. Reducing this rich input language to the simpler form recognized by the core engine is carried out by a *preprocessor* module.

The preprocessor performs some basic normalization of atoms, so that the search engine only has to deal with a restricted set of predicates. It expands ternary *if-then-else* constructs over math terms. Finally, it uses a standard linear-time, satisfiability preserving translation to transform the formula (including the remaining *if-then-else* constraints on the boolean level) into conjunctive normal form.

The preprocessor partitions the set of mathematical atoms into disjoint clusters. Two atoms belong to the same cluster if they share a variable. Instead of one single LA solver, each cluster is handled by a separate LA solver. (This can be seen as the simplest form of combination of theories, where there are no interface variables.)

In addition, the preprocessor uses a form of *static learning* to add some satisfiability preserving clauses on atoms occurring in ϕ, that help to prune the search space in the boolean level (e.g., $\neg(t_1 - t_2 \leqslant 3) \vee \neg(t_2 - t_3 \leqslant 5) \vee (t_1 - t_3 \leqslant 9)$).

Finally, the preprocessor performs the propositional abstraction of the formula, instantiating the mapping between mathematical atoms and the corresponding propositions used in boolean solving.

Boolean solver

The propositional abstraction of the mathematical formula produced by the preprocessor is given to the boolean satisfiability solver, extended to implement the MathSAT algorithm described in Section 3.2. This solver is built upon the MiniSat solver [14], from which it inherits conflict-driven learning and backjumping, restarts [33,8,18], optimized boolean constraint propagation based on the two-watched literal scheme, and a variant of the VSIDS splitting heuristics [22]. The communication with the theory solvers is carried out through an interface (similar to the one in [17]) that passes assigned literals, consistency queries and backtracking commands, and receives back answers to the queries, mathematical conflict sets and implied literals.

The boolean solver has been extended to handle some options relevant when dealing with mathematical formulas. For instance, MathSAT inherits MiniSat's feature of periodically discarding some of the learned clauses to prevent explosion of the formula size. However, clauses generated by theory-driven learning and forward deduction mechanisms are never discarded, as a default option, since they may have required a lot of theory reasoning. As a second example, it is possible to initialize the VSIDS heuristics weights of literals so that either boolean or theory atoms are preferred as splitting choices early in the MathSAT search.

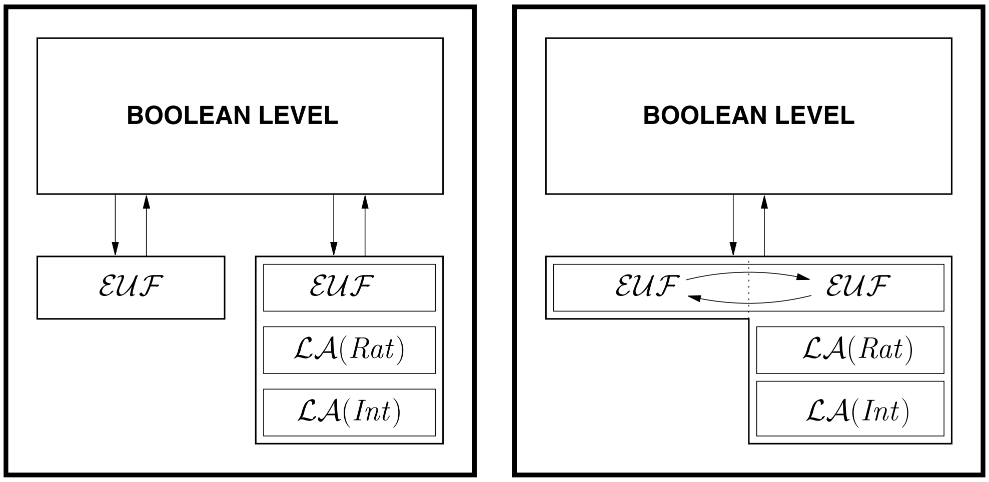

Fig. 8. An architectural view of MathSAT (left) and MathSAT ($\mathcal{EUF}$,$\mathcal{LA}$) (right).

The theory solver for $\mathcal{EUF}$

The implementation of the $\mathcal{EUF}$ solver reuses some of the data structures of the theorem prover E [29] to store and process terms and atoms. It is based on the congruence-closure algorithm described in [24]. It is fully incremental and resettable, and is able to construct conflict sets, discover implied literals, and explanations. Thus, the $\mathcal{EUF}$ solver is e_{ij}-deduction complete.

The theory solver for $\mathcal{LA}$

The solver for $\mathcal{LA}$ is a complex structure, where various submodules are integrated: the submodule for $\mathcal{DL}$ uses the Bellman–Ford algorithm to handle difference logic constraints; the $\mathcal{LA}(Rat)$ submodule employs the simplex algorithm to deal with linear arithmetic over the reals; $\mathcal{LA}(Int)$ is tackled by a (resource-bounded) branch-and-cut procedure, which tries to find an integer solution in the solution space over the rationals; the final check is performed via Fourier-Motzkin elimination.

Besides being incremental and resettable, the most distinguishing feature is the idea of *layering*: cheaper theories are used before more expensive theories. First, the whole assignment is given to a solver for $\mathcal{EUF}$, which treats all arithmetic operations as uninterpreted functions. Since many unsatisfiable assignments turn out to be already unsatisfiable with purely equational reasoning, this often prevents more complex solvers to be called. Only if the current assignment is satisfiable in $\mathcal{EUF}$ will the solver for $\mathcal{LA}(Rat)$ be called to check for satisfiability over the rational numbers. Finally, if the problem is over the integers, the expensive solver for $\mathcal{LA}(Int)$ will only be called if the problem turned out to be satisfiable over the rationals.

5.2. Implementing the Delayed Theory Combination Schema

The Delayed Theory Combination Schema has been implemented in the MathSAT framework for both $\mathcal{EUF} \cup \mathcal{LA}(Rat)$ and $\mathcal{EUF} \cup \mathcal{LA}(Int)$. In a straightforward implementation, boolean assignments to the interface equalities would be given both to the solver for $\mathcal{EUF}$ and the solver for $\mathcal{LA}$, thereby forcing a consistent assignment to these interface equalities. In MathSAT, the solver for $\mathcal{EUF}$ and the first layer of the solver for $\mathcal{LA}$ (which is a solver for $\mathcal{EUF}$ treating arithmetic operations as uninterpreted functions) have been merged. (see Fig. 8, right). Hence, in practice, the interface equalities are handed only to the lower (arithmetical) layers of the $\mathcal{LA}$-solver if they are consistent with the purely $\mathcal{EUF}$-part of the current assignment.

Filtering theory atoms

The following (optional) techniques have been devised for *filtering* the atoms that are passed to the theory solvers for T-satisfiability tests. The idea is that the theory solvers are given smaller but sufficient sets of constraints that can be hopefully solved faster. Note that the techniques below *only apply to literals that are not interface equalities.* This is because interface equalities form the consistency bridge between two theories.

Pure Literal Filtering. Assume that a Σ-literal l is propositionally pure in the conjunctive normal form formula Φ, meaning that $\neg l^p$ does not appear in Φ^p. Now, if a truth assignment β^p is such that $\neg l^p \in \beta^p$ propositionally satisfies Φ, then so will $(\beta^p \setminus \{\neg l^p\}) \cup \{l^p\}$. Therefore, if l^p is assigned to false in the current partial truth assignment, then $\neg l$ does not have to be passed to the theory solver.

Theory-Deduced Literal Filtering. Assume that a clause $C = (l_i \vee \cdots \vee l_n)$ of Σ-literals is entailed by the underlying theory T, i.e. the conjunction $\neg l_1 \wedge \cdots \wedge \neg l_n$ is T-unsatisfiable. (Such clauses result from theory-driven learning, deduction, and static learning.) If (i) the current truth assignment β^p contains the literals $\neg l_1^p, \ldots, \neg l_{n-1}^p, l_n^p$, (ii) each $\neg l_1, \ldots, \neg l_{n-1}$ has been given to the theory solver, and (iii) l_n^p is forced to true by unit propagation on the clause C, then l_n does not have to be given to the theory solver as *prop2fol* $(\beta^p \setminus \{l_n^p\})$ is T-satisfiable iff *prop2fol* (β^p) is.

Management of interface equalities

The most important points to be emphasized are related to the management of the interface atoms.

- The preprocessor now also purifies the formula: the interface variables x_i are identified, and the interface equalities e_{ij} are added to the theory solvers. Note that the interface equalities are added even if they do not appear in the formula.
- Intuitively, it seems a good idea to delay the activation of e_{ij} atoms; in order to achieve this objective, we instructed the SAT solver not to branch on them until no other choice is left. This is done by suitably initializing the activity vector controlling the VSDIS splitting strategy [22]. In practice, this choice may cause a significant speedup, as shown in Section 7.3.
- Once the search finds a (both T_1-consistent and T_2-consistent) truth assignment satisfying ϕ^p, we are not done: to guarantee correctness, we need an assignment also for those e_{ij}'s which still do not have a value. This is provided by the SAT solver used in MathSAT, which constructs total[8] assignments over the propositional variables that are declared.

[8] See Note 3.

- Before any new split, the current (partial) assignment is checked for T_1- and T_2-consistency, and the procedure backtracks if it is found inconsistent. In this way, the SAT solver enumerates *total* assignments on e_{ij}'s only if strictly necessary.
- Depending on the search, it is possible that e_{ij} are given values not only by branching, but also by boolean constraint propagation on learned clauses, or even by theory deduction. In fact, the e_{ij} interface equalities are also fed into the congruence closure solver, which also implements forward deduction [5] and therefore is able to assign forced truth values (e.g., to conclude the truth of $x_1 = x_2$ from the truth of $x = x_1$, $y = x_2$, and $x = y$). This reduces branching at boolean level, and limits the delay of combination between the theories.
- When an interface equality e_{ij} is involved in a conflict, it is treated like the other atoms by the conflict-driven splitting heuristic: i.e., its branching weight is increased, and it becomes more likely to be split upon. Furthermore, the conflict clause is learned, and it is thus possible to prevent incompatible configurations between interface atoms and the other propositions.
- The initial value attempted for each unassigned e_{ij} is false. If x_i and x_j were in the same equivalence class because of equality reasoning, then e_{ij} had already been forced to true by equality reasoning. Therefore, we are guaranteed that setting e_{ij} to false will not result in expensive merging of equivalence classes, even though conflicts can result in the arithmetic solver. In practice, this may cause a significant speedup, as shown is Section 7.3.

6. Related work

The Nelson–Oppen integration schema was first proposed in [23], whereas the Shostak method was proposed independently in [30]. Recently, a number of papers have been proposed to clarify the issues related to the combination of Shostak theories by studying their relationship with the Nelson–Oppen schema. We refer the reader to [26] for a synthesis of Nelson–Oppen and Shostak approaches. The notion of extended canonizer for Shostak theories is discussed in [27,32,26].

The approach presented in this paper can be seen as lifting a nondeterministic variant of the Nelson–Oppen integration schema to the case of formulae with arbitrary boolean structure. We exploit this similarity in order to prove the correctness of our approach. With respect to the approaches described above, however, where the schema is mostly defined as a formal tool, we also focus on the practical issues, and show how to reach an efficient implementation by leveraging state-of-the-art techniques in boolean search and in *SMT*, and on theory layering (i.e., cheaper reasoning first, and more often).

In fact, to our knowledge, the integration schema described in this paper has never been proposed elsewhere as the basis for a solver. The most closely related systems, which are able to deal with combinations of theories using variants of $\mathsf{Bool}+\mathsf{no}(T_1, T_2)$, are the following. CVCLite [10,3] is a library for checking validity of quantifier-free first-order formulas over several interpreted theories, including $\mathcal{LA}(Rat)$, $\mathcal{LA}(Int)$, $\mathcal{EUF}$, and arrays, replacing the older tools SVC and CVC. Verifun [15] is a similar tool, supporting domain-specific procedures for $\mathcal{EUF}$, $\mathcal{LA}$, and the theory of arrays. Zapato [6] is a counterexample-driven abstraction refinement tool, able to decide the combination of $\mathcal{EUF}$ and a specific fragment of $\mathcal{LA}(Int)$. ICS [19,16] is able to deal with uninterpreted function symbols and a large variety of theories, including arithmetic, tuples, arrays, and bit-vectors. ICS [27,32] somewhat departs from the $\mathsf{Bool}+\mathsf{no}(T_1, T_2)$ schema, by mixing Shostak approach (by merging canonizers for individual theories into a global canonizer), with Nelson–Oppen integration schema (to deal with non-Shostak theories).

Other approaches implementing Bool+T for a single theory are [38,9,17]. The work in [17] proposes a formal characterization of the Bool+T approach, and an efficient instantiation to a decision procedure for $\mathcal{EUF}$ (based on an incremental and resettable congruence closure algorithm [24], which is also implemented in MATHSAT). Despite its generality for the case of a single theory, the approach is bound to the Bool+T schema, and requires an integration between theory solvers to deal with $SMT(T_1 \cup T_2)$.

A different approach to *SMT* is the "eager" reduction of a decision problem for T to propositional SAT. This approach has been successfully pioneered by UCLID [40,31], a tool incorporating a decision procedure for $\mathcal{EUF}$, arithmetic of counters, separation predicates, and arrays. This approach leverages on the accuracy of the encodings and on the effectiveness of propositional SAT solvers, and performs remarkably well for certain theories. However, it sometimes suffers from a blow-up in the encoding to propositional logic, see for instance a comparison in [17] on $\mathcal{EUF}$ problems. The bottleneck is even more evident in the case of more expressive theories such as $\mathcal{LA}$ [34,35], and in fact UCLID gives up the idea of a fully eager encoding [20]. The most relevant subcase for this approach is $\mathcal{DL} \cup \mathcal{EUF}$, which is addressed in [34,28]. Unfortunately, it was impossible to make a comparison due to the unavailability of the benchmarks (only the benchmarks after Ackermann expansion were made available to us).

7. Experimental evaluation

In this section we evaluate experimentally the Delayed Theory Combination approach. We first describe the set of test cases used for the experiments (Section 7.1); then, we discuss a comparison between the DTC implementation of MATHSAT and some competitor systems (Section 7.2); finally, in Section 7.3 we evaluate the impact of the proposed DTC optimizations on the overall performance of MATHSAT.

7.1. Description of the tests

There is a general lack of test suites on $\mathcal{EUF} \cup \mathcal{LA}$. For instance, the tests in [28] were made available only after Ackermann expansion, so as to drop the $\mathcal{EUF}$ component. We also analyzed the tests in the regression suite for CVCLITE [10], but they turned out to be extremely easy.

We defined the following benchmarks suites.

Modular Arithmetic. This suite simulates arithmetic operations (succ, pred, sum) modulo N. Some basic variables range between 0 and N, and the problem is to decide the satisfiability of (the negation of) known mathematical facts. Most problems are unsatisfiable. The test suite comes in two versions: one purely $\mathcal{EUF}$, where the behavior of arithmetic operations is tabled (e.g. $s(0) = 1, \dots, s(N) = 0$); one in $\mathcal{EUF} \cup \mathcal{LA}$, where each arithmetic operation has also a characterization via $\mathcal{LA}$ and conditional expressions (e.g. $p(x, y) = if\ (x + y < N)\ then\ x + y\ else\ x + y - N$) taking into account overflows.

Random Problems. We developed a random generator for SMT($\mathcal{EUF} \cup \mathcal{LA}(Rat)$) problems. The propositional structure is a 3-CNF; the atoms can be either fully propositional, equalities between two terms, or a comparison between a term and a numerical constant. A basic term is an individual variable between $x_1, \ldots, x_n$; a compound terms x_i, with $i > n$, is either the application of an uninterpreted function symbol (e.g. $f(x_{j_1}, \ldots, x_{j_n})$), or a ternary linear polynomial with random coefficients to previously defined terms. The generator depends on the *coupling*: high coupling increases the probability that a subterm of the term being generated is a compound term rather than a variable. A problem class is identified by the number of variables and clauses, and the coupling; for each configuration of the parameters we defined 20 random samples.

Hash. The suite contains some problems over hash tables. They are modeled with a combination of uninterpreted function symbols and integer ranges. Some constraints specify the right behavior of the table, ensuring different integer values for different locations; then a property involving multiple nesting of different hash tables is checked (e.g. $1 \leqslant x < n \rightarrow hash_1(hash_2(x + x)) = hash_1(hash_2(2 * x))$, where x is an integer variable, and n is half of the size of the tables).

The test suite is parametrized in the number of the hash tables and in their size. We provide both satisfiable and unsatisfiable instances.

7.2. Comparison with other systems

We ran the implementation of MathSAT based on Delayed Theory Combination (MathSAT-Dtc hereafter) against the alternative implementation in MathSAT based on Ackermann expansion (MathSAT-Ack hereafter), and against the competitor tools ICS (v.2.0) and CVCLite (v.1.1.0). (We also tried to use the unstable version of CVCLite, which is somewhat more efficient, but it was unable to run the tests due to internal errors.) We ran the tools over all the benchmarks suites, for a total of more than 3800 test formulae for each tool.

All the experiments were run on a 2-processor INTEL Xeon 3 GhZ machine with 4 Gb of memory, running Linux RedHat 7.1 (only one processor was allowed to run for each run). The time limit was set to 1800 seconds and the memory limit to 500 MB. An executable version of MathSAT and the source files of all the experiments performed in the paper are available at [21].

The overall results are reported in Fig. 9. Each of the columns show the comparison between MathSAT-Dtc and MathSAT-Ack, CVCLite, and ICS, respectively; the rows correspond to the different test suites. MathSAT-Dtc dominates MathSAT-Ack on all the problems except the ones on Modular Arithmetic on $\mathcal{EUF}$. MathSAT-Dtc dominates CVCLite on all the problems. The comparison with ICS is limited to problems in the first three benchmarks corresponding to the the first three rows; since ICS is incomplete over the integers, it returns incorrect results in the Hash suite (which is on $\mathcal{EUF} \cup \mathcal{LA}(Int)$, with a non-trivial $\mathcal{LA}(Int)$ component). In the first row, MathSAT-Dtc generally outperforms ICS. On the second row, MathSAT-Dtc behaves better than ICS on part of the problems, while it times out on others. On the problems in the third row, ICS is faster than MathSAT, even though the suite turns out to be relatively easy for both systems (most tests were run within one second).

7.3. Impact of the different solutions/strategies

We also evaluated the impact on the overall performance of the specific optimizations that we devised for DTC. We compared the default version of MathSAT-Dtc (the one used for the experiments in Section 7.2) against three different configurations, obtained by disabling *one optimization*

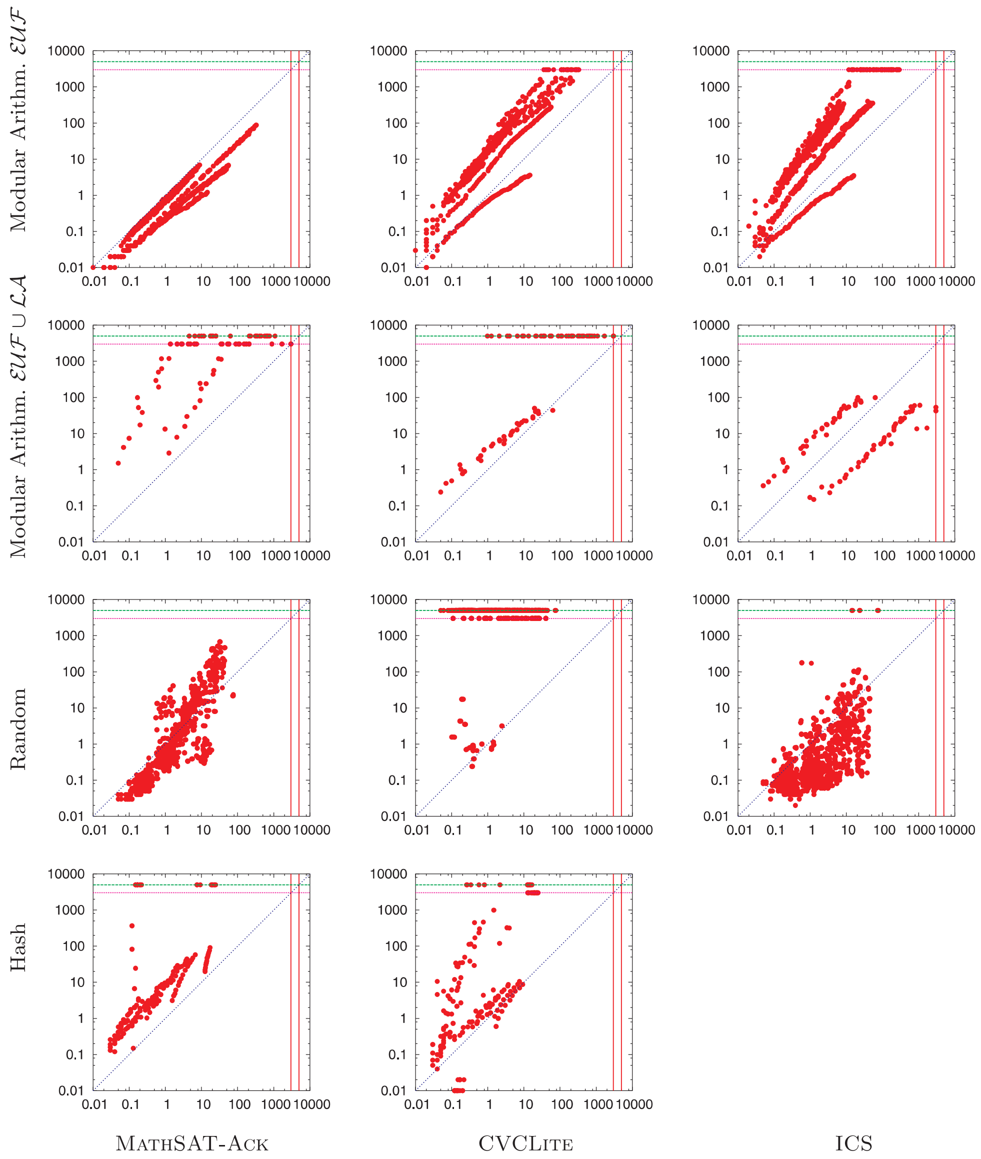

Fig. 9. Execution time ratio: the X and Y axes report MathSAT-Dtc and each competitor's times, respectively (logarithmic scale). A dot above the diagonal means a better performance of MathSAT-Dtc and viceversa. The two uppermost horizontal lines and the two rightmost vertical lines represent, respectively, out-of-memory (higher) or time-out (lower).

at a time (in other words, each version that has been tested differs from the default version only with respect to one of the optimizations). The configurations we tested are, respectively:

No delayed e_{ij} splitting. The configuration obtained by disabling the delayed activation of interface equalities; with this configuration, the system is more likely to split on interface atoms in earlier stages of the search.

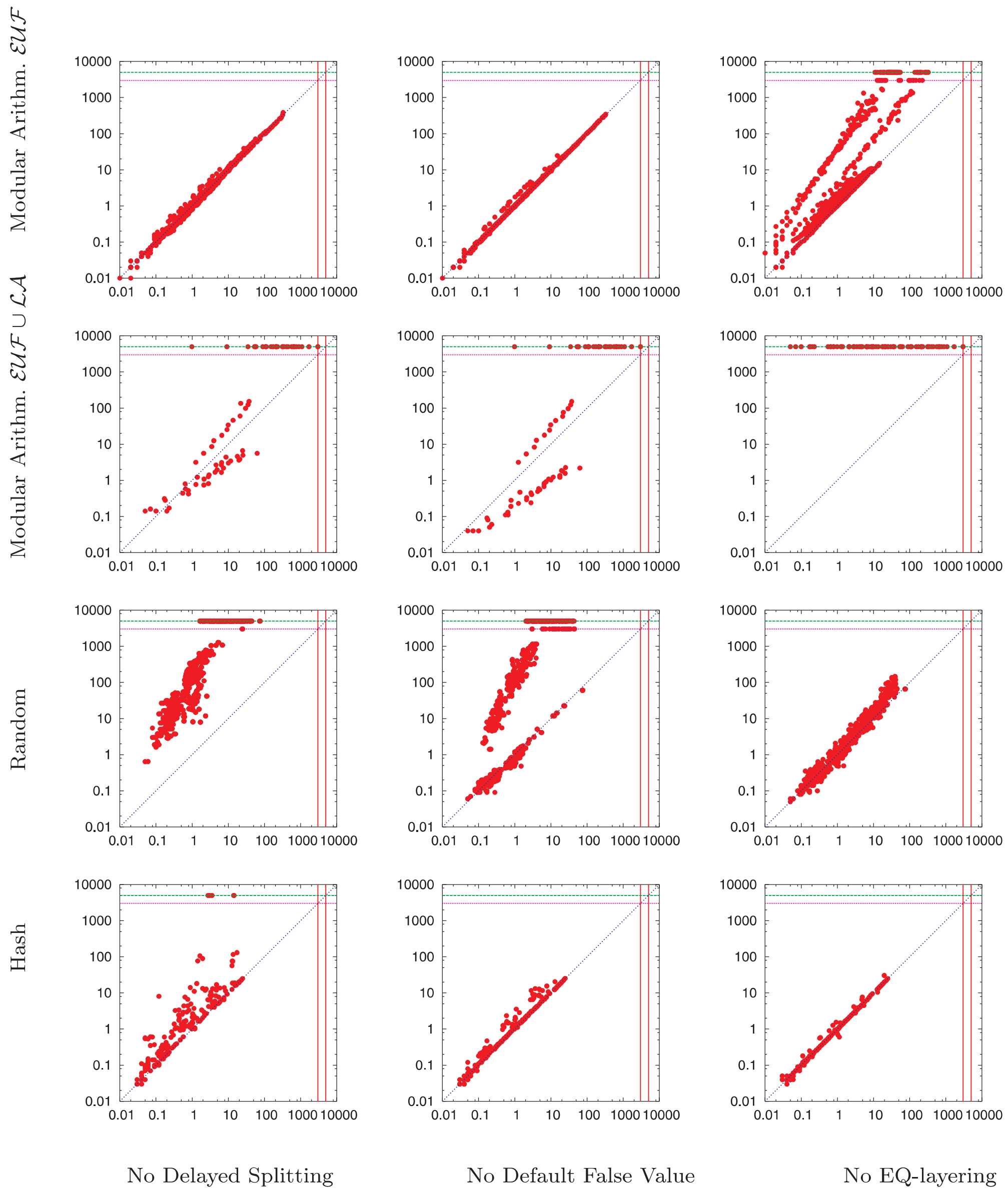

Fig. 10. Execution time ratio: the X and Y axes report the default version of MathSAT-Dtc and its variations' times, respectively (logarithmic scale). A dot above the diagonal means a better performance of the default version and viceversa. The two uppermost horizontal lines and the two rightmost vertical lines represent, respectively, out-of-memory (higher) or time-out (lower).

No default e_{ij} false value. The configuration obtained by disabling the choice of using "false" as a default split value for each unassigned interface equality; due to the way the splitting mechanism is realized in MiniSat, this actually amounts to force the initial value of unassigned interface equalities to "true". This means that the $\mathcal{EUF}$ reasoner is more likely to deal with positive equalities, and thus merge equivalence classes; on the other hand, notice that negated equalities may induce splitting in $\mathcal{LA}$.

No EQ-layering. The configuration of MathSAT-Dtc without EQ-layering. In this configuration the solver for $\mathcal{EUF}$ is not used as the first layer in the solver for $\mathcal{LA}$. However, deduction on interface equalities is still carried out by the solver for $\mathcal{EUF}$.

We ran the different configurations over all the benchmarks suites, with the same time and memory limits as in Section 7.2.

The overall results are reported in Fig. 10. The columns show the comparison between MathSAT-Dtc and its variations, in the order described above; the rows correspond to the different test suites. In general, the optimizations introduce at most marginal degrade in performance; in fact, in many cases they result in dramatic speed-ups. It is also interesting to notice that the different optimizations seem to interact positively, i.e. the elimination of any of them yields significant performance degradation in some classes of problems.

8. Conclusions and future work

In this paper, we have proposed a new approach to the combination of theories for the $SMT(T_1 \cup T_2)$ problem. The approach privileges the interaction between the boolean component and each of the theories, and delays the interaction between each of the theories. This approach is very expressive (e.g., it nicely encompasses the case of non-convex theories), and much simpler to understand and analyze; each of the solvers can be implemented and optimized without taking into account the others; furthermore, in our approach it is possible to investigate the different degrees of deduction which can be carried out by the theory solvers.

We have implemented the approach in the MathSAT solver [5], for the case of the combination of $\mathcal{LA}$ and $\mathcal{EUF}$, both in the convex case of rationals, and the non-convex case of integers. The experimental evaluation shows that the use of dedicated solvers for $\mathcal{EUF}$, and the control of the split over interface atoms, are crucial to make the approach efficient. In fact, the system compares positively with state-of-the-art competitors, sometimes with dramatic gains in performance.

In the future, we will proceed along the following directions. On one side, we will increase the performance of the solver, in particular in the case of problems in $\mathcal{LA}(Rat)$. During the evaluation, most of the time is spent in arithmetic computations, and several bottlenecks have already been identified. Furthermore, we will investigate the idea of on-the-fly purification, and more accurate heuristics to control splitting. On the other side, we will investigate the idea of incremental satisfiability, and the integration of MathSAT with domain-specific verification flows based on inductive reasoning (e.g., verification of timed and hybrid systems, and of RTL circuits). Finally, we plan to widen the scope of applicability of our approach to combination of theories for which a generalization of the Nelson–Oppen schema is required. In fact, there have been some attempts to lift the Nelson–Oppen schema to non-disjoint combinations [41] or to disjoint combinations of non-stably infinite theories [39]. Such schemas generalize the Nelson–Oppen approach by guessing identifications over a larger set of variables than that of interface variables (this is necessary for completeness). Our approach can be easily adapted to consider interface atoms over such an extended set of variables. This allows us to obtain rapid prototypes of these schemas which, by now, have only been interesting from a theoretical point of view, given the difficulties of a reasonable implementation of the guessing phase. This paves the way to conduct experimental evaluations of how SMT-techniques scale up for a wider range of combinations of theories.

References

[1] A. Armando, C. Castellini, E. Giunchiglia, M. Maratea, A SAT-based decision procedure for the boolean combination of difference constraints, in: Proceedings of SAT'04, 2004.

[2] W. Ackermann, Solvable Cases of the Decision Problem, North Holland, Amsterdam, 1954.

[3] C.L. Barrett, S. Berezin, CVC Lite: a new implementation of the cooperating validity checker, in: Proceedings of CAV'04, Lecture Notes in Computer Sciences, vol. 3114, Springer, Berlin, 2004.

[4] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, R. Sebastiani, Efficient satisfiability modulo theories via delayed theory combination, in: Proceedings of CAV 2005, Lecture Notes in Computer Sciences, vol. 3576, Springer, Berlin, 2005.

[5] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, R. Sebastiani, An incremental and layered procedure for the satisfiability of linear arithmetic logic, in: Proceedings of TACAS'05, Lecture Notes in Computer Sciences, vol. 3440, Springer, Berlin, 2005.

[6] T. Ball, B. Cook, S.K. Lahiri, L. Zhang, Zapato: automatic theorem proving for predicate abstraction refinement, in: CAV, Lecture Notes in Computer Sciences, vol. 3114, 2004.

[7] C.W. Barret, D.L. Dill, A. Stump, A generalization of Shostak's method for combining decision procedures, in: Proceedings of FROCOS'02, 2002.

[8] R.J. Bayardo Jr, R.C. Schrag, Using CSP look-back techniques to solve real-world SAT instances, in: Proceedings of AAAI/IAAI'97, AAAI Press, 1997, pp. 203–208.

[9] S. Cotton, E. Asarin, O. Maler, P. Niebert, Some progress in satisfiability checking for difference logic, in: Proceedings of FORMATS-FTRTFT 2004, 2004.

[10] CVC, CVCL, SVC. Available from: <http://verify.stanford.edu/{CVC,CVCL,SVC}>.

[11] N. Dershowitz, J.-P. Jouannaud, Rewrite systems, in: in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier and MIT Press, Amsterdam/Cambridge, MA, 1990, pp. 243–320.

[12] D. Deharbe, S. Ranise, Light-Weight theorem proving for debugging and verifying units of code, in: Proceedings of SEFM'03, IEEE Computer Society Press, Sliver Spring, MD, 2003.

[13] H.B. Enderton, A Mathematical Introduction to Logic, Academic Press, New York, 1972.

[14] N. Eén, N. Sörensson, An extensible SAT-solver, in: Theory and Applications of Satisfiability Testing (SAT 2003), Lecture Notes in Computer Sciences, vol. 2919, Springer, Berlin, 2004, pp. 502–518.

[15] C. Flanagan, R. Joshi, X. Ou, J.B. Saxe, Theorem proving using lazy proof explication, in: Proceedings of the CAV'03, Lecture Notes in Computer Sciences, vol. 2725, Springer, Berlin, 2003, pp. 355–367.

[16] J.-C. Filliâtre, S. Owre, H. Rueß, N. Shankar, ICS: integrated canonizer and solver, in: Proceedings of CAV'01, Lecture Notes in Computer Sciences, vol. 2102, 2001, pp. 246–249.

[17] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, C. Tinelli, DPLL(T): fast decision procedures, in: Proceedings of CAV'04, Lecture Notes in Computer Sciences, vol. 3114, Springer, Berlin, 2004, pp. 175–188.

[18] C. Gomes, B. Selman, H. Kautz, Boosting combinatorial search through randomization, in: Proceedings of the Fifteenth National Conference on Artificial Intelligence, American Association for Artificial Intelligence, 1998, pp. 431–437.

[19] ICS. Available from: <http://www.icansolve.com>.

[20] D. Kroening, J. Ouaknine, S. Seshia, O. Strichman, Abstraction-based satisfiability solving of Presburger arithmetic, in: Proceedings of CAV'04, Lecture Notes in Computer Sciences, vol. 3114, Springer, Berlin, 2004, pp. 308–320.

[21] MathSAT. Available from: <http://mathsat.itc.it>.

[22] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: engineering an efficient SAT solver, in: Proceedings of the DAC'01, ACM, 2001, pp. 530–535.

[23] G. Nelson, D.C. Oppen, Simplification by cooperating decision procedures, ACM Trans. Programm. Lang. Syst. 1 (2) (1979) 245–257.

[24] R. Nieuwenhuis, A. Oliveras, Congruence closure with integer offsets, in: Proceedings of the 10th LPAR, number 2850 in LNAI, Springer, Berlin, 2003, pp. 77–89.
[25] D.C. Oppen, Complexity, convexity and combinations of theories, Theor. Comput. Sci. 12 (1980) 291–302.
[26] S. Ranise, C. Ringeissen, D.-K. Tran, Nelson–Oppen, Shostak, and the extended canonizer: a family picture with a newborn, in: Proceedings of the ICTAC'04, 2004.
[27] H. Rueß, N. Shankar, Deconstructing Shostak, in: Proceedings of the LICS'01, IEEE Computer Society, 2001, pp. 19–28.
[28] S.A. Seshia, R.E. Bryant, Deciding quantifier-free Presburger formulas using parameterized solution bounds, in: Proceedings of the LICS'04, 2004.
[29] S. Schulz, E—A Brainiac Theorem Prover, AI Commun. 15 (2/3) (2002) 111–126.
[30] R.E. Shostak, Deciding combinations of theories, J. ACM 31 (1984) 1–12.
[31] S.A. Seshia, S.K. Lahiri, R.E. Bryant, A hybrid SAT-based decision procedure for separation logic with uninterpreted functions, in: Proceedings of the DAC'03, 2003.
[32] N. Shankar, H. Rueß, Combining Shostak theories, in: RTA, vol. 2378, 2002.
[33] J.P.M. Silva, K.A. Sakallah, GRASP—a new search algorithm for satisfiability, in: Proceedings of the ICCAD'96, ACM, 1996, pp. 220–227.
[34] O. Strichman, S. Seshia, R. Bryant, Deciding separation formulas with SAT, in: Proceedings of the Computer Aided Verification, (CAV'02), Lecture Notes in Computer Sciences, Springer, Berlin, 2002.
[35] O. Strichman, On solving Presburger and linear arithmetic with SAT, in: Proceedings of Formal Methods in Computer-Aided Design (FMCAD 2002), Lecture Notes in Computer Sciences, Springer, Berlin, 2002.
[36] C. Tinelli, M. Harandi, A new correctness proof of the Nelson–Oppen combination procedure, in: Proceedings of the FROCOS'96, 1996.
[37] C. Tinelli, C. Ringeissen, Unions of non-disjoint theories and combinations of satisfiability procedures, Theor. Comput. Sci. 290 (1) (2003) 291–353.
[38] TSAT++. Available from: <http://www.ai.dist.unige.it/Tsat>.
[39] C. Tinelli, C.G. Zarba, Combining non-stably infinite theories, J. Automated Reasoning (2005) (to appear).
[40] UCLID. Available from: <http://www-2.cs.cmu.edu/~uclid>.
[41] C.G. Zarba, Combining sets with integers, in: A. Armando (Ed.), Proceedings of the FROCOS'02, Lecture Notes in Computer Sciences, vol. 2309, Springer, Berlin, 2002, pp. 103–116.