Efficient Satisfiability Modulo Theories via Delayed Theory Combination *

Marco Bozzano[1], Roberto Bruttomesso[1], Alessandro Cimatti[1], Tommi Junttila[2], Silvio Ranise[3], Peter van Rossum[4], and Roberto Sebastiani[5]

[1] ITC-IRST, Povo, Trento, Italy. {bozzano,bruttomesso,cimatti}@itc.it
[2] Helsinki University of Technology, Finland. Tommi.Junttila@tkk.fi
[3] LORIA and INRIA-Lorraine, Villers les Nancy, France. Silvio.Ranise@loria.fr
[4] Radboud University Nijmegen, The Netherlands. petervr@sci.kun.nl
[5] DIT, Università di Trento, Italy. roberto.sebastiani@unitn.it

Abstract. The problem of deciding the satisfiability of a quantifier-free formula with respect to a background theory, also known as Satisfiability Modulo Theories (*SMT*), is gaining increasing relevance in verification: representation capabilities beyond propositional logic allow for a natural modeling of real-world problems (e.g., pipeline and RTL circuits verification, proof obligations in software systems).

In this paper, we focus on the case where the background theory is the combination $T_1 \cup T_2$ of two simpler theories. Many *SMT* procedures combine a boolean model enumeration with a decision procedure for $T_1 \cup T_2$, where conjunctions of literals can be decided by an integration schema such as Nelson-Oppen, via a structured exchange of interface formulae (e.g., equalities in the case of convex theories, disjunctions of equalities otherwise).

We propose a new approach for $SMT(T_1 \cup T_2)$, called Delayed Theory Combination, which does not require a decision procedure for $T_1 \cup T_2$, but only individual decision procedures for T_1 and T_2, which are directly integrated into the boolean model enumerator. This approach is much simpler and natural, allows each of the solvers to be implemented and optimized without taking into account the others, and it nicely encompasses the case of non-convex theories. We show the effectiveness of the approach by a thorough experimental comparison.

1 Introduction

Many practical verification problems can be expressed as satisfiability problems in decidable fragments of first-order logic. In fact, representation capabilities beyond propositional logic enable a natural modeling of real-world problems (e.g., pipeline and RTL circuits verification, proof obligations in software systems).

* 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.

The field has been devoted a lot of interest and has recently acquired the name *Satisfiability Modulo Theories* (*SMT*). *SMT* can be seen as an extended form of propositional satisfiability, where propositions are constraints in a specific theory. A prominent approach which underlies several systems (e.g., MATHSAT [16,6], DLSAT [7], DPLL(T) [13], TSAT++ [28,2], ICS [14,11], CVCLITE [8,4], haRVey [9]), 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.

The above schema, which we denote as $\mathsf{Bool}{+}T$, is also followed when the background theory T turns out to be the combination $T_1 \cup T_2$ of two simpler theories — for instance, Equality and Uninterpreted Functions ($\mathcal{E}$) and Linear Arithmetic ($\mathcal{LA}$). The decision procedure to decide a combination of literals in T is typically based on an integration schema such as Nelson-Oppen (NO) [18] (we denote the resulting schema as $\mathsf{Bool}{+}\mathsf{no}(T_1,T_2)$), starting from decision procedures for each T_i, and combining them by means of a structured exchange of interface formulae.

In this paper, we propose a new approach for the $SMT(T_1 \cup T_2)$ problem, called *Delayed Theory Combination*. The main idea is to avoid the integration schema between T_1 and T_2, and tighten the connection between each T_i and the boolean level. While the truth assignment is being constructed, it is checked for consistency with respect to each theory in isolation. This can be seen as constructing two (possibly inconsistent) partial models for the original formula; the "merging" of the two partial models is enforced, on demand, since the solver is requested to find a complete assignment to the *interface equalities*.

We argue that this approach, denoted as $\mathsf{Bool}{+}T_1+T_2$, has several advantages over $\mathsf{Bool}{+}\mathsf{no}(T_1,T_2)$. First, the whole framework is much simpler to analyze and implement; 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 $\mathsf{Bool}{+}T_i$. 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 for efficiency and which is in fact hampering the search – or too difficult to implement. Third, the framework nicely encompasses the case of non-convex theories: 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. We experimentally show that our approach is competitive and often superior to the other state of the art approaches based on Nelson-Oppen integration.

This paper is structured as follows. We first present some background and overview the $\mathsf{Bool}{+}T$ procedure in Sect. 2. Then we discuss the $T_1 \cup T_2$ case by means of the Nelson-Oppen combination schema in Sect. 3. We present our approach $\mathsf{Bool}{+}T_1{+}T_2$ in Sect. 4. Then we describe the implementation of $\mathsf{Bool}{+}T_1{+}T_2$ for the case of $\mathcal{LA} \cup \mathcal{E}$ in Sect. 5 and some related work in Sect. 6. We discuss the experimental evaluation in Sect. 7. Finally, we draw some conclusions and discuss some future work in Sect. 8.

2 Satisfiability Modulo Theories

Satisfiability Modulo a Theory is the problem of checking the satisfiability of a quantifier-free (or ground) first-order formula with respect to a given first-order theory T. Theories of interest are, e.g., the theory of difference logic $\mathcal{DL}$, where constraints have the form

function Bool$+T$ (ϕ: *quantifier-free formula*)
$\mathcal{A}^p \longleftarrow \mathit{fol2prop}(\mathit{Atoms}(\phi))$
$\phi^p \longleftarrow \mathit{fol2prop}(\phi)$
while Bool-$\mathit{satisfiable}(\phi^p)$ **do**
$\beta^p \longleftarrow \mathit{pick\_total\_assign}(\mathcal{A}^p, \phi^p)$
$(\rho, \pi) \longleftarrow T\text{-}\mathit{satisfiable}(\mathit{prop2fol}(\beta^p))$
if $\rho = \mathsf{sat}$ **then return** sat
$\phi^p \longleftarrow \phi^p \wedge \neg \mathit{fol2prop}(\pi)$
end while
return unsat
end function

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

$x - y \leq c$; the theory $\mathcal{E}$ of equality and uninterpreted functions, whose signature contains a finite number 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 over the rationals (or, equivalently, over the reals), hereafter denoted with $\mathcal{LA}(\mathit{Rat})$; the quantifier-free fragment of Linear Arithmetic over the integers, hereafter denoted with $\mathcal{LA}(\mathit{Int})$.

Figure 1 presents Bool$+T$, a (much simplified) decision procedure for $\mathit{SMT}(T)$. The function *Atoms* takes a ground formula ϕ and returns the set of atoms which occurs in ϕ. We use the notation ϕ^p to denote the *propositional abstraction* of ϕ, which is formed by the function *fol2prop* that maps propositional variables to themselves, ground atoms into fresh propositional variables, and is homomorphic w.r.t. boolean operators and set inclusion. The function *prop2fol* is the inverse of *fol2prop*. We use β^p to denote a propositional assignment, i.e. a conjunction (a set) of propositional literals. The idea underlying the algorithm is that the truth assignments for the propositional abstraction of ϕ are enumerated and checked for satisfiability in T. The procedure either concludes satisfiability if one such model is found, or returns with failure otherwise. The function *pick_total_assign* returns a total assignment to the propositional variables in ϕ^p, that is, it assigns a truth value to all variables in $\mathcal{A}^p$. The function 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 $\pi \subseteq \beta$ is a T-unsatisfiable set, called a *theory conflict set*. We call the negation of a conflict set, a *conflict clause*.

The algorithm is a coarse abstraction of the ones underlying TSAT++, MathSAT, DLSAT, DPLL(T), CVCLite, haRVey, and ICS. The test for satisfiability and the extraction of the corresponding truth assignment are kept separate in this description only for the sake of simplicity. In practice, enumeration is carried out on *partial assignments*, by means of efficient boolean reasoning techniques, typically by means of a DPLL-algorithm (but see also [9] for an approach based on BDDs). Additional improvements are *early pruning*, i.e., partial assignments which are not T-*satisfiable* are pruned (since no refinement can be T-*satisfiable*); theory conflicts discovered by the theory solver can be passed as conflict clauses to the boolean solver, and trigger non-chronological backjumping; such conflict clauses can also be *learned*, and induce the discovery of more compact learned clauses; finally, *theory deduction* can be used to reduce the search space by explicitly returning truth values for unassigned literals, as well as constructing/learning implications. The interested reader is pointed to [6] for details and further references.

3 $SMT(T_1 \cup T_2)$ via Nelson-Oppen Integration

In many practical applications of $SMT(T)$, the background theory is a combination of two theories T_1 and T_2. For instance, $\mathcal{DL}$ and $\mathcal{E}$; $\mathcal{LA}(Rat)$ and $\mathcal{E}$; $\mathcal{LA}(Int)$ and $\mathcal{E}$; $\mathcal{LA}(Int)$, $\mathcal{E}$ and the theory of arrays. Many recent approaches to $SMT(T_1 \cup T_2)$ (e.g. CVCLITE, ICS) rely on the adaptation of the Bool+T schema, by instantiating T-*satisfiable* with some decision procedure for the satisfiability of $T_1 \cup T_2$, typically based on the Nelson-Oppen integration schema (see Figure 2, left). In the following, we briefly recall the most relevant issues pertaining to the combination of decision procedures. (For a more complete discussion we refer the reader to [20].) [1]

Let Σ_1 and Σ_2 be two disjoint signatures, and let T_i be a Σ_i-theory for $i = 1,2$. A $\Sigma_1 \cup \Sigma_2$-term t is an *i-term* if it is a variable or it has the form $f(t_1,...,t_n)$, where f is in Σ_i and $n \geq 0$ (notice that a variable is both a 1-term and a 2-term). A non-variable subterm s of an i-term is *alien* if s is a j-term, and all superterms of s 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. A literal is i-pure if it contains only i-pure terms. A formula is said to be *pure* iff every literal occurring in the formula is i-pure for some $i \in \{1,2\}$. The process of *purification* maps any formula ϕ into an equisatisfiable pure formula ϕ' by introducing new variables and definitions to rename non-pure/alien terms. Especially, if ϕ is a conjunction of literals, then ϕ' can be written as $\phi_1 \wedge \phi_2$ s.t. each ϕ_i is a conjunction of i-pure literals. In the following, we call *interface variables* of a pure formula ϕ' the set of all variables $c_1,\ldots,c_n \in Var(\phi')$ that occur in both 1-pure and 2-pure literals in ϕ', and we define e_{ij} as the *interface equality* $c_i = c_j$.

A Σ-theory T is called *stably-infinite* iff for any T-satisfiable Σ-formula ϕ, there exists a model of T whose domain is infinite and which satisfies ϕ. A *Nelson-Oppen* theory is a stably-infinite theory which admits a satisfiability algorithm. E.g., $\mathcal{E}$, $\mathcal{DL}$, $\mathcal{LA}(Rat)$, and $\mathcal{LA}(Int)$ are all Nelson-Oppen theories. A conjunction Γ of Σ-literals is *convex* in a Σ-theory T iff for any disjunction $\bigvee_{i=1}^{n} x_i = y_i$ (where x_i, y_i are variables) 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. Note that, while $\mathcal{E}$ and $\mathcal{LA}(Rat)$ are convex theories, $\mathcal{LA}(Int)$ is not: e.g., given the variables x,x_1,x_2, the set $\{x_1 = 1, x_2 = 2, x_1 \leq x, x \leq x_2\}$ entails $x = x_1 \vee x = x_2$ but neither $x = x_1$ nor $x = x_2$.

Given two signature-disjoint Nelson-Oppen theories T_1 and T_2, the Nelson-Oppen combination schema [18], in the following referred to as $\mathsf{no}(T_1,T_2)$, allows one to solve the satisfiability problem for $T_1 \cup T_2$ (i.e. the problem of checking the $T_1 \cup T_2$-satisfiability of conjunctions of $\Sigma_1 \cup \Sigma_2$-literals) by using the satisfiability procedures

[1] Notice that the Nelson-Oppen schema of Figure 2, left, is a simplified one. In actual implementations (e.g., CVCLITE, ICS) more than two theories can be handled at a time, and the interface equalities are exchanged between theory solvers by exploiting sophisticated techniques (see e.g. [10] for details).

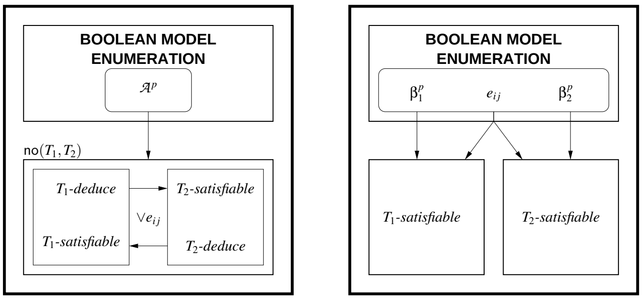

Fig. 2. The different schemas for $SMT(T_1 \cup T_2)$

for T_1 and T_2. The procedure is basically a structured interchange of information inferred from either theory and propagated to the other, until convergence is reached. The schema requires the exchange of information, the kind of which depends on the *convexity* of the involved theories. In the case of convex theories, the two solvers communicate to each other interface equalities. In the case of non-convex theories, the $\mathsf{no}(T_1, T_2)$ schema becomes more complicated, because the two solvers need to exchange *arbitrary disjunctions of interface equalities*, which have to be managed within the decision procedure by means of case splitting and of *backtrack* search.

We notice that the ability to carry out deductions is often crucial for efficiency: each solver must be able to derive the (disjunctions of) interface equalities e_{ij} entailed by its current facts ϕ. When this capability is not available, it can be replaced by "guessing" followed by a satisfiability check with respect to T_i.

Example 1. Let T_1 be $\mathcal{E}$ and T_2 be $\mathcal{LA}(Int)$, and consider the following *SMT* problem for the purified formula, $V = \{x, w_1, w_2\}$ being the set of interface variables:

$$\phi = \neg(f(x) = f(w_1)) \wedge (A \leftrightarrow \neg(f(x) = f(w_2))) \;\wedge\; 1 \leq x \wedge x \leq 2 \wedge w_1 = 1 \wedge w_2 = 2.$$

Suppose we first assign the boolean variable A to true (branch 1), so that ϕ simplifies into a conjunction of literals $\phi_1 \wedge \phi_2$, s.t., $\phi_1 = \neg(f(x) = f(w_1)) \wedge \neg(f(x) = f(w_2))$ and $\phi_2 = 1 \leq x \wedge x \leq 2 \wedge w_1 = 1 \wedge w_2 = 2$. Then the $\mathsf{no}(T_1, T_2)$ schema runs as follows:

1. The literals of ϕ_1 are processed, T_1-satisfiability is reported, and no equality is derived.
2. The literals of ϕ_2 are processed, T_2-satisfiability is reported, and the disjunction $x = w_1 \vee x = w_2$ is returned.
3. The disjunction induces a case-splitting; first, $x = w_1$ is passed to the solver for T_1:
   (a) $\phi_1 \wedge x = w_1$ is T_1-unsatisfiable, since $\neg(f(x) = f(w_1)) \wedge x = w_1$ is;
   then, $x = w_2$ is passed to the satisfiability procedure for T_1:
   (b) $\phi_1 \wedge x = w_2$ is T_1-unsatisfiable, since $\neg(f(x) = f(w_2)) \wedge x = w_2$ is.
   The T_1-solver may be able to return the conflict clauses C_1: $\neg(x = w_1) \vee f(x) = f(w_1)$ and C_2: $\neg(x = w_2) \vee f(x) = f(w_2)$ to the boolean solver, which learns them to drive future search.
4. $\mathsf{no}(T_1, T_2)$ returns the $T_1 \cup T_2$-unsatisfiability of $\phi_1 \wedge \phi_2$, and the procedure backtracks.

Then we assign A to false (branch 2), so that ϕ_1 becomes $\neg(f(x) = f(w_1)) \wedge (f(x) = f(w_2))$. Hence the $\mathsf{no}(T_1, T_2)$ combination schema reruns steps 1, 2, and 3(a) as in branch 1. (Notice that, if the conflict clause C_1 has been generated, then $\neg(x = w_1)$

function Bool$+T_1+T_2$ (ϕ_i: *quantifier-free formula*)

1 $\phi \longleftarrow purify(\phi_i)$

2 $\mathcal{A}^p \longleftarrow fol2prop(Atoms(\phi) \cup E(interface\_vars(\phi)))$

3 $\phi^p \longleftarrow fol2prop(\phi)$

4 **while** Bool-*satisfiable* (ϕ^p) **do**

5 $\beta_1^p \wedge \beta_2^p \wedge \beta_e^p = \beta^p \longleftarrow pick\_total\_assign(\mathcal{A}^p, \phi^p)$

6 $(\rho_1, \pi_1) \longleftarrow T_1$-*satisfiable* $(prop2fol(\beta_1^p \wedge \beta_e^p))$

7 $(\rho_2, \pi_2) \longleftarrow T_2$-*satisfiable* $(prop2fol(\beta_2^p \wedge \beta_e^p))$

8 **if** $(\rho_1 = \mathsf{sat} \wedge \rho_2 = \mathsf{sat})$ **then return** sat **else**

9 **if** $\rho_1 = \mathsf{unsat}$ **then** $\phi^p \longleftarrow \phi^p \wedge \neg fol2prop(\pi_1)$

10 **if** $\rho_2 = \mathsf{unsat}$ **then** $\phi^p \longleftarrow \phi^p \wedge \neg fol2prop(\pi_2)$

11 **end while**

12 **return** unsat

end function

Fig. 3. A simplified view of the Delayed Theory Combination procedure for $SMT(T_1 \cup T_2)$

is added to branch 2 by the boolean solver, so that step 2 generates only $x = w_2$, and hence step 3(a) is skipped.) Then, $x = w_2$ is passed to the satisfiability procedure for T_1, which states that $\phi_1 \wedge x = w_2$ is T_1-satisfiable, and that no new interface equalities are deducible. Hence $\phi_1 \wedge \phi_2$ in branch 2 is $T_1 \cup T_2$-satisfiable, so that the original formula ϕ is $T_1 \cup T_2$-satisfiable.

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

We propose a new approach to $SMT(T_1 \cup T_2)$, which 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 (see Fig. 2, right), and each of the theory solvers works in isolation, without direct exchange of information. Their mutual consistency is ensured by conjoining the purified formula with a suitable formula, containing only the interface equalities e_{ij}, even if these do not occur in the original problem; such a formula encodes all possible equivalence relations over the interface variables in the purified formula. The enumeration of assignments includes not only the atoms in the formula, but also the interface atoms 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): the synchronization between the theory reasoners is delayed until the e_{ij}'s are associated a value. We denote this schema as Bool$+T_1+T_2$.

A simplified view of the algorithm is presented in Fig. 3. Initially (lines 1–3), the formula is purified, the interface variables c_i are identified by *interface_vars*, the interface equalities e_{ij} are created by E and added to the set of propositional symbols $\mathcal{A}^p$, and the propositional abstraction ϕ^p of ϕ is created. Then, the main loop is entered (lines 4–11): while ϕ^p is propositionally satisfiable (line 4), we select a satisfying truth assignment β^p (line 5). We remark that truth values are associated not only to atoms in ϕ, but also to the e_{ij} atoms, even though they do not occur in ϕ. β^p is then (implicitly) separated into $\beta_1^p \wedge \beta_e^p \wedge \beta_2^p$, where *prop2fol*$(\beta_i^p)$ is a set of *i*-pure literals and *prop2fol*(β_e^p) is a set of e_{ij}-literals. The relevant part of β^p are checked for consistency against each theory (lines 6–7); T_i*-satisfiable* (β) returns a pair (ρ_i, π_i), where ρ_i is unsat iff β is unsatisfiable in T_i, and sat otherwise. 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, ϕ^p is strengthened to exclude truth assignments which may fail in the same way (line 9–10), and the loop is resumed. Unsatisfiability is returned (line 12) when the loop is exited without having found a model.

To see why Bool$+T_1+T_2$ is a decision procedure for $SMT(T_1 \cup T_2)$, let us first consider the case where ϕ is a conjunction of literals. In this case, we claim that the correctness and completeness of Bool$+T_1+T_2$ reduces to that of a nondeterministic version of Nelson-Oppen combination schema (see e.g. [27]). The traditional, deterministic Nelson-Oppen schema relies on the exchange of entailed interface equalities, i.e. discovering that e_{ij} is entailed by the set of literals ϕ modulo the theory T_i. In the nondeterministic case, the same deduction is simulated by "guessing" that e_{ij} holds, and then checking whether $\phi \wedge \neg e_{ij}$ is T_i-unsatisfiable. Similar reasoning applies in the dual case where we "guess" that e_{ij} is false. In Bool$+T_1+T_2$, the selection of the truth assignment on line 5 corresponds to guessing a truth value for each of the e_{ij}, while the calls to T_i*-satisfiable* of lines 6 and 7 check the consistency of this guess with respect to each T_i. According to [27], $T_1 \cup T_2$-satisfiability can be concluded when both checks return sat. Otherwise, another guess should be attempted: this is carried out by strengthening the formula with the conflict clause (lines 9–10), and selecting a different total assignment to the e_{ij}. This result can be generalized to the case when ϕ is an arbitrary formula. We consider that ϕ is satisfiable iff there exists a satisfying assignment to its literals, which is also $T_1 \cup T_2$-satisfiable. It is not difficult to see that the set of assignments enumerated by the algorithm is the same set obtained by enumerating the assignments of Atoms(ϕ), and then extending it with a complete assignment over the e_{ij}.

For lack of space, the algorithm is described in Fig. 3 at a high level of abstraction. In practice, enumeration is carried out by means of a DPLL-based SAT engine, and all the optimizations discussed for Bool$+T$ can be retained. For a thorough discussion of these issues, we refer the reader to an extended version of this paper [5]. Here, we only emphasize the role of theory deduction, where a call to T_i*-satisfiable*, when satisfiable, can return in π_i a set of theory deductions (i.e. theory-justified implications, which may in turn force truth values on unassigned literals, thus further constraining the boolean search space).

Example 2. Consider the formula and the situation of Example 1. As before, we first assign A to true (branch 1), so that $\neg(f(x) = f(w_2))$. 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.

1. Choosing $w_1 = w_2$ causes a T_2-inconsistency be revealed by early-pruning calls to the theory solvers, so that the conflict clause C_3: $\neg(w_1 = 1) \vee \neg(w_2 = 2) \vee \neg(w_1 = w_2)$ is learned, and the SAT solvers backtracks to $\neg(w_1 = w_2)$, which does not cause inconsistency.
2. Similarly, choosing $x = w_1$ causes a T_1-inconsistency, the conflict clause C_1 of example 1 is learned, and the SAT solvers backtracks to $\neg(x = w_1)$, which does not cause inconsistency.
3. Similarly, choosing $x = w_2$ causes a T_1-inconsistency, the conflict clause C_2 of example 1 is learned, and the SAT solvers backtracks to $\neg(x = w_2)$.
4. $\neg(x = w_1)$ and $\neg(x = w_2)$ cause a T_2-inconsistency, so that branch 1 is closed.

Then we assign A to false (branch 2), so that $f(x) = f(w_2)$. Hence $\neg(x = w_1)$ and $\neg(w_1 = w_2)$ are immediately assigned by unit-propagation on C_1 and C_3. Thus, after splitting on $x = w_2$ we have a satisfying assignment.

Notice that (i) when a *partial* assignment on e_{ij}'s is found unsatisfiable under some T_i (e.g., $w_1 = w_2$ in branch 1, step 1), then all its *total* extensions are T_i-unsatisfiable, so that there is no need for further boolean search on the other e_{ij}'s. Therefore techniques like early pruning, learning and theory deduction allow for restricting the search on *partial* assignments; (ii) the extra boolean component of search caused by the non-convexity of $\mathcal{LA}(Int)$ has been merged into the top-level boolean search, so that it can be handled efficiently by the top-level DPLL procedure.

The following observations indicate what are 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 $\mathsf{Bool}+T_i$. All features from the DPLL framework such as early pruning, theory driven backjumping and learning, deduction, and split control can be used.

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 in fact $SMT(T_1 \cup T_2)$ problems may feature complex interactions between the boolean level and each of the T_i.

Multiple Theories The DTC approach can be easily extended to handle the combination of $n > 2$ component theories. We only need to dispatch each satisfiability procedure the conjunction of pure literals extended with a total assignment on the interface equalities e_{ij} and return the satisfiability of the formula if all report satisfiability. In case some of the procedures report unsatisfiability, the conflict sets are added to the formula and a new propositional assignment is considered. We see no practical difficulty to implement the DTC schema for $n > 2$ theories, although we have not yet investigated this experimentally.

Deduction The NO schema relies on theory solvers being deduction-complete, that is, being able to always infer all the (disjunctions of) e_{ij}'s which are entailed in the theory by the input set of theory literals. However, 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. 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.

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 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.

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 conflict obtained in one 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.

As possible drawbacks, we notice that DTC requires the *whole* formula to be purified, and the upfront introduction of $O(n^2)$ interface constraints e_{ij}. 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 can help to limit the increase in the search. On the whole, we expect that the DTC schema will make it easier the task of extending *SMT* tools to handle combination of theories while ensuring a high-degree of efficiency. In fact, the DTC approach does not need dedicated mechanisms to exchange selected formulae nor to handle non-convex theories, thereby greatly simplifying the implementation task. On the one hand, we believe that systems based on our approach can be made competitive with more traditional systems on theories where deduction of entailed facts can be efficiently done, by adapting techniques developed for SAT solvers. On the other hand, the DTC approach offers a flexible framework to explore the different trade-offs of deduction for theories where deriving entailed facts is computationally expensive.

5 Delayed Theory Combination in Practice: MATHSAT $(\mathcal{E} \cup \mathcal{LA})$

We implemented the Delayed Theory Combination schema presented in the previous section in MATHSAT [6]. MATHSAT is an SMT solver for each of the theories $\mathcal{DL}$, $\mathcal{LA}(Rat)$, $\mathcal{LA}(Int)$, and $\mathcal{E}$. Furthermore, it is also an SMT solver for $(\mathcal{E} \cup \mathcal{LA}(Rat))$ and for $(\mathcal{E} \cup \mathcal{LA}(Int))$, where uninterpreted symbols are eliminated by means of Ackermann's expansion [1]. MATHSAT is based on an enhanced version of the Bool+T schema (see [6] for further details).

We instantiated the Delayed Theory Combination schema to deal with $\mathcal{E} \cup \mathcal{LA}(Rat)$ and with $\mathcal{E} \cup \mathcal{LA}(Int)$. During preprocessing, the formula is purified, the interface variables c_i are identified, and the interface equalities e_{ij} are added to the solver. The most important points to be emphasized are related to the management of the interface atoms:

- in order to delay the activation of e_{ij} atoms, we instructed the SAT solver not to branch on them until no other choice is left; (by suitably initializing the activity vector controlling the VSIDS splitting strategy [17]);
- once the search finds a truth assignment that satisfies ϕ^p and is also T_1- and T_2-satisfiable, we are not done: to guarantee correctness, we need an assignment also for those e_{ij}'s that still do not have a value. This is provided by the SAT solver used in MATHSAT, which constructs total assignments over the propositional variables that are declared;

– before any new split, the current (partial) assignment is checked for T_1- and T_2-satisfiability, and the procedure backtracks if it is found unsatisfiable. 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 [6] and therefore is able to assign forced truth values (e.g., to conclude the truth of $c_1 = c_2$ from the truth of $x = c_1$, $y = c_2$, and $x = y$). This reduces branching at boolean level, and limits the delay of combination between the theories;
– when e_{ij} is involved in a conflict, it is treated like the other atoms by the conflict-driven splitting heuristic: 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 c_i and c_j were in the same equivalence class because of equality reasoning, then e_{ij} had already been forced to true by equality reasoning. Thus c_i and c_j belong to different equivalence classes in the congruence closure solver and setting e_{ij} to false will not result in expensive merging of equivalence classes nor otherwise change the state of the solver. However, conflicts can result in the arithmetic solver.

6 Related Work

To our knowledge, the integration schema we describe in this paper has not been previously proposed elsewhere. Most closely related are the following systems, which are able to deal with combination of theories, using variants of $\mathsf{Bool}+\mathsf{no}(T_1, T_2)$. CVCLite [8,4] is a library for checking validity of quantifier-free first-order formulas over several interpreted theories, including $\mathcal{LA}(Rat)$, $\mathcal{LA}(Int)$, $\mathcal{E}$, and arrays, replacing the older tools SVC and CVC. Verifun [12] is a similar tool, supporting domain-specific procedures for $\mathcal{E}$, $\mathcal{LA}$, and the theory of arrays. Zapato [3] is a counterexample-driven abstraction refinement tool, able to decide the combination of $\mathcal{E}$ and a specific fragment of $\mathcal{LA}(Int)$. ICS [14,11] is able to deal with uninterpreted function symbols and a large variety of theories, including arithmetic, tuples, arrays, and bit-vectors. ICS [21,24] 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's theories).

Other approaches implementing $\mathsf{Bool}+T$ for a single theory are [28,7,13]. The work in [13] proposes a formal characterization of the $\mathsf{Bool}+T$ approach, and an efficient instantiation to a decision procedure for $\mathcal{E}$ (based on an incremental and backtrackable congruence closure algorithm [19], which is also implemented in MathSAT). Despite its generality for the case of a single theory, the approach is bound to the $\mathsf{Bool}+T$ 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 [29, 23], a tool incorporating a decision procedure for $\mathcal{E}$, 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 [13] on $\mathcal{E}$ problems. The bottleneck is even more evident in the case of more expressive theories such as $\mathcal{LA}$ [26, 25], and in fact UCLID gives up the idea of a fully eager encoding [15]. The most relevant subcase for this approach is $\mathcal{DL} \cup \mathcal{E}$, which is addressed in [22]. Unfortunately, it was impossible to make a comparison due to the unavailability of the benchmarks (only the benchmarks after Ackermann's expansion were made available to us).

7 Experimental Evaluation

We ran the implementation of MATHSAT with Delayed Theory Combination (hereafter called MATHSAT-DTC) against the alternative implementation based on Ackermann's expansion (hereafter called MATHSAT-ACK), and 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). Unfortunately, there is a general lack of test suites on $\mathcal{E} \cup \mathcal{LA}$ available. For instance, the tests in [22] were available only after Ackermann's expansion, so that the $\mathcal{E}$ component has been removed. We also analyzed the tests in the regression suite for CVCLITE [8], but they turned out to be extremely easy. We defined the following benchmarks suites.

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

Random Problems We developed a random generator for SMT($\mathcal{E} \cup \mathcal{LA}(\mathit{Rat})$) 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. We denote a class of problem as RND(vartype, n, clauses, coupling); for each configuration of the parameters we defined 20 random samples.

Hash The suite contains some problems over hash tables modeled as integer-valued bijective functions over finite sets of integers.

We ran the four tools on over 3800 test formulae. The experiments were run on a 2-processor INTEL Xeon 3 GhZ machine with 4 Gb of memory, running Linux RedHat 7.1. The time limit was set to 1800 seconds (only one processor was allowed to run for

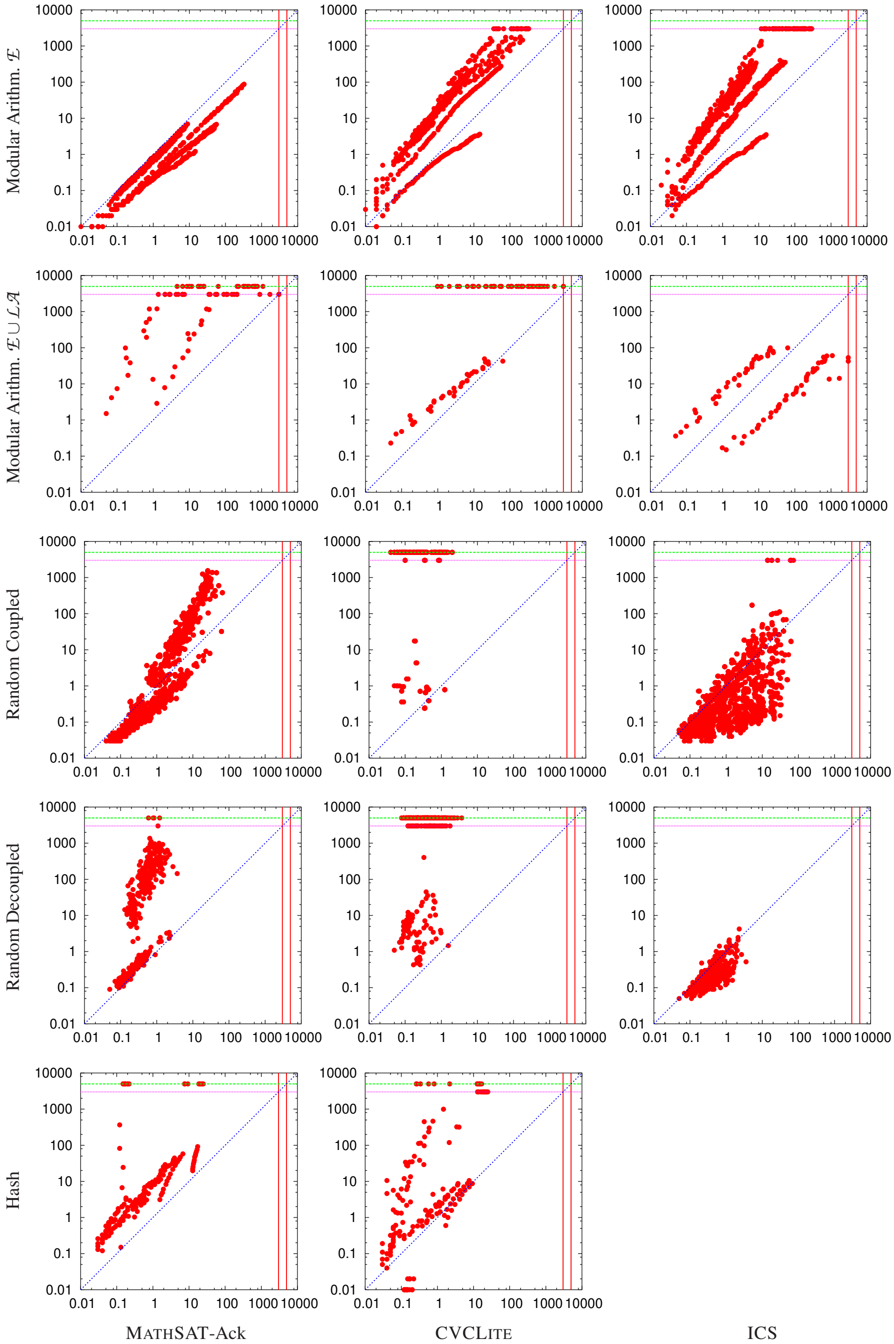

Fig. 4. Execution time ratio: the X and Y axes report MATHSAT-DTC and each competitor's times, respectively (logarithmic scale). A dot above the diagonal means a better performance of MATHSAT-DTC and viceversa. The two uppermost horizontal lines and the two rightmost vertical lines represent, respectively, out-of-memory (higher) or timed-out (lower)

each run) 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 [16].

The results are reported in Fig. 4. The columns show the comparison between MATHSAT-DTC and MATHSAT-ACK, CVCLITE, ICS; the rows correspond to the different test suites. MATHSAT-DTC dominates CVCLITE on all the problems, and MATHSAT-ACK on all the problems except the ones on Modular Arithmetic on $\mathcal{E}$. [2] The comparison with ICS is limited to problems in $\mathcal{E} \cup \mathcal{LA}(Rat)$, i.e. the first four rows (the Hash suite is in $\mathcal{E} \cup \mathcal{LA}(Int)$ and ICS, being incomplete over the integers, returns incorrect results). In the first row, MATHSAT-DTC generally outperforms ICS. On the second row, MATHSAT-DTC behaves better than ICS on part of the problems, and worse on others. In the third and fourth rows, MATHSAT-DTC is slower than ICS on simpler problems, but more effective on harder ones (for instance, it never times out); this is more evident in the third row, due to the fact that the problems in the fourth row are simpler (most of them were run within one second).

8 Conclusions and Future Work

In this paper we have proposed a new approach for tackling the problem of Satisfiability Modulo Theories (SMT) for the combination of theories. Our approach is based on delaying the combination, and privileging the interaction between the boolean component and each of the theories. This approach is much simpler to analyze and implement; each of the solvers can be implemented and optimized without taking into account the others; furthermore, our approach does not rely on the solvers being deduction-complete, and it nicely encompasses the case of non-convex theories. We have implemented the approach in the MATHSAT [6] solver for the combination of the theories of Equality and Uninterpreted Functions ($\mathcal{E}$) and Linear Arithmetic, over the rationals ($\mathcal{LA}(Rat)$) and the integers ($\mathcal{LA}(Int)$), and we have shown its effectiveness experimentally.

As future work, we plan to further improve MATHSAT by investigating new ad hoc optimizations for $\mathcal{LA}(Rat) \cup \mathcal{E}$ and $\mathcal{LA}(Int) \cup \mathcal{E}$. In particular, the most evident limitation of the approach presented in this paper is the upfront introduction of interface equalities. We believe that this potential bottleneck could be avoided by means of a lazy approach, which will be the objective of future research. We also want to provide a more extensive experimental evaluation on additional sets of benchmarks. Finally, we plan to apply our framework for the verification of RTL circuit designs, where the combination of $\mathcal{LA}(Int)$ and $\mathcal{E}$ is essential for representing complex designs.

References

1. W. Ackermann. *Solvable Cases of the Decision Problem.* North Holland Pub. Co., 1954.
2. A. Armando, C. Castellini, E. Giunchiglia, and M. Maratea. A SAT-based Decision Procedure for the Boolean Combination of Difference Constraints. In *Proc. SAT'04*, 2004.

[2] The tests for CVCLITE on the "random coupled" benchmark (3rd row, 2nd column in Fig. 4) are not complete, because on nearly all samples CVCLITE produced either a time-out or a out-of-memory, so that we could not complete on time the whole run on the 2400 formulas of the benchmark.

3. T. Ball, B. Cook, S.K. Lahiri, and L. Zhang. Zapato: Automatic Theorem Proving for Predicate Abstraction Refinement. In *CAV 2004*, volume 3114 of *LNCS*. Springer, 2004.
4. C. Barrett and S. Berezin. CVC Lite: A New Implementation of the Cooperating Validity Checker. In *CAV 2004*, volume 3114 of *LNCS*. Springer, 2004.
5. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, S. Ranise, P.van Rossum, and R. Sebastiani. Efficient Theory Combination via Boolean Search. Technical Report T05-04-02, ITC-IRST, 2005.
6. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. An incremental and Layered Procedure for the Satisfiability of Linear Arithmetic Logic. In *TACAS 2005*, volume 3440 of *LNCS*. Springer, 2005.
7. S. Cotton, E. Asarin, O. Maler, and P. Niebert. Some Progress in Satisfiability Checking for Difference Logic. In *FORMATS/FTRTFT 2004*, volume 3253 of *LNCS*. Springer, 2004.
8. CVC, CVCLITE and SVC. `http://verify.stanford.edu/{CVC,CVCL,SVC}`.
9. D. Deharbe and S. Ranise. Light-Weight Theorem Proving for Debugging and Verifying Units of Code. In *Proc. SEFM'03*. IEEE Computer Society Press, 2003.
10. D. Detlefs, G. Nelson, and J.B. Saxe. Simplify: A Theorem Prover for Program Checking. Technical Report HPL-2003-148, HP Laboratories, 2003.
11. J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated Canonizer and Solver. In *CAV 2001*, volume 2102 of *LNCS*. Springer, 2001.
12. C. Flanagan, R. Joshi, X. Ou, and J.B. Saxe. Theorem Proving using Lazy Proof Explication. In *CAV 2003*, volume 2725 of *LNCS*. Springer, 2003.
13. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *CAV 2004*, volume 3114 of *LNCS*. Springer, 2004.
14. ICS. `http://www.icansolve.com`.
15. D. Kroening, J. Ouaknine, S. A. Seshia, , and O. Strichman. Abstraction-Based Satisfiability Solving of Presburger Arithmetic. In *CAV 2004*, volume 3114 of *LNCS*. Springer, 2004.
16. MATHSAT. `http://mathsat.itc.it`.
17. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. DAC'01*, pages 530–535. ACM, 2001.
18. G. Nelson and D.C. Oppen. Simplification by Cooperating Decision Procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, 1979.
19. R. Nieuwenhuis and A. Oliveras. Congruence Closure with Integer Offsets. In *LPAR 2003*, number 2850 in LNAI. Springer, 2003.
20. S. Ranise, C. Ringeissen, and D.-K. Tran. Nelson-Oppen, Shostak, and the Extended Canonizer: A Family Picture with a Newborn. In *ICTAC 2004*, volume 3407 of *LNCS*, 2005.
21. H. Rueß and N. Shankar. Deconstructing Shostak. In *Proc. LICS'01*, pages 19–28. IEEE Computer Society, 2001.
22. S.A. Seshia and R.E. Bryant. Deciding Quantifier-Free Presburger Formulas Using Parameterized Solution Bounds. In *Proc. LICS'04*. IEEE, 2004.
23. S.A. Seshia, S.K. Lahiri, and R.E. Bryant. A Hybrid SAT-Based Decision Procedure for Separation Logic with Uninterpreted Functions. In *DAC 2003*. ACM, 2003.
24. N. Shankar and H. Rueß. Combining Shostak Theories. In *RTA 2002*, volume 2378 of *LNCS*. Springer, 2002.
25. O. Strichman. On Solving Presburger and Linear Arithmetic with SAT. In *FMCAD 2002*, volume 2517 of *LNCS*. Springer, 2002.
26. O. Strichman, S. Seshia, and R. Bryant. Deciding separation formulas with SAT. In *CAV 2002*, volume 2404 of *LNCS*. Springer, 2002.
27. C. Tinelli and M. Harandi. A New Correctness Proof of the Nelson-Oppen Combination Procedure. In *Proc. FroCos'96*. Kluwer Academic Publishers, 1996.
28. TSAT++. `http://www.ai.dist.unige.it/Tsat`.
29. UCLID. `http://www-2.cs.cmu.edu/~uclid`.