Science of Computer Programming 77 (2012) 518–532

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Combining decision procedures by (model-)equality propagation☆

Diego Caminha B. de Oliveira [a,b,c], David Déharbe [d,*], Pascal Fontaine [a,b,c]

[a] Université de Nancy, Nancy, France
[b] INRIA, Nancy, France
[c] LORIA, Nancy, France
[d] Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte, Natal – RN, Brazil

ARTICLE INFO

Article history:
Received 31 January 2009
Received in revised form 15 April 2010
Accepted 16 April 2010
Available online 27 April 2010

Keywords:
Formal verification
Automated theorem proving
SMT solving
Combination of decision procedures

ABSTRACT

Formal methods in software and hardware design often generate formulas that need to be validated, either interactively or automatically. Among the automatic tools, SMT (Satisfiability Modulo Theories) solvers are particularly suitable to discharge such proof obligations, as their input language is equational logic with symbols from various useful decidable fragments such as uninterpreted symbols, linear arithmetic, and usual data-structures like arrays or lists. In this paper, we present an approach to combine decision procedures and propositional solvers into an SMT-solver, based not only on the exchange of deducible equalities between decision procedures, but also on the generation of model equalities by decision procedures. This extends nicely the classical Nelson–Oppen combination procedure in a simple platform to smoothly combine convex and non-convex theories. We show the soundness and completeness of this approach using an original abstract framework to represent and reason about SMT-solvers. We then describe an algorithmic translation of this method, implemented in the kernel of the veriT solver (Bouton et al. (2009)) [9].

1. Introduction

The application of formal methods to the design of computing systems often results in the generation of verification conditions that need to be proved in order to guarantee the correctness of the result. Such verification conditions express properties of models or relations between models and may be expressed in a wide range of logics: from propositional to higher order logics, but also process algebras and temporal logics. Hence the level of automation for verification in a specific formalism is tightly dependent on the availability of tools to support reasoning in such logics.

The work described in this paper addresses the verification of satisfiability modulo theories (SMT) of quantifier-free formulas, i.e. verification conditions expressed in a first-order logic using symbols from a combination of theories, such as uninterpreted functions, fragments of integer or real arithmetic, set and array theories, etc. This applies to a number of verification applications, e.g. the application of formal program transformations such as refinement [25] or refactoring laws [11], verification of refinement properties in *posit-and-prove* software engineering efforts [1,2], or static analysis of annotations in design-by-contract languages [24]. Even verification efforts in more expressive logics often require proving lemmas that may be tackled by SMT-solvers (see for instance [8]). This work is therefore a contribution towards the construction of more efficient and trustworthy automated theorem provers based on the SMT-solving approach

☆ The research presented in this paper has been partially supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq grant 573964/2008-4, and by the ANR project DECERT.

* Corresponding author.
E-mail addresses: Diego.Caminha@loria.fr (D.C.B. de Oliveira), deharbe@gmail.com, david.deharbe@pq.cnpq.br (D. Déharbe), Pascal.Fontaine@loria.fr (P. Fontaine).

0167-6423/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2010.04.003

(see [6] for a thorough review of state-of-the-art SMT-solving techniques). The goal of the our research is to develop such SMT-solving techniques, and provide effective implementations thereof.

SMT-solvers can, for example, handle formulas such as

$$x \leq y \ \wedge \ y \leq x + f(x) \ \wedge \ P(h(x) - h(y)) \ \wedge \ \neg P(0) \ \wedge \ f(x) = 0 \tag{1}$$

which contains linear arithmetic on real numbers $(0, +, -, \leq)$, and uninterpreted symbols (P, h, f). SMT-solvers use decision procedures for the disjoint languages (for instance, congruence closure for uninterpreted symbols [28], and simplex for linear arithmetic) and combine them to build a decision procedure for the union of the languages. The combination of decision procedures works either through some guessing, or through the exchange of information between the decision procedures. The decision procedures thus combined are composed with a Boolean reasoning engine. In the classical schema for combining decision procedures, the information exchanged between the decision procedures is a set of disjunctions of equalities. Handling such disjunctions requires complex and costly case splitting. In the special case of convex theories[1] (see definition in Section 2.1) though, exchanging only equalities instead of disjunctions of equalities is enough to ensure the completeness of the combination and the combination schema is much simpler.

This paper is based on results first presented in [15]. We present a new generic combination approach that consists in exchanging only equalities, instead of disjunctions thereof. The exchanged equalities are either deduced like in the classical schema, or assumed by generalizing models. We show that, even in the general case (i.e. with non-convex theories), this approach is complete, thanks to the cooperation with the propositional reasoning engine of the SMT-solver. This combination schema is suitable for any decision procedure capable of finding models; fortunately many decision procedures inherently have this capability. The combination approach described in the paper is the core algorithm of the SMT-solver veriT [9]. The complexity of combination schema is of course bound to the usual theoretical limits (e.g. NP-complete in the cases studied in [33,32]), but it gives good results in many practical situations.

This paper also presents an abstract framework to describe and reason about SMT-solvers built using different kinds of cooperation between decision procedures. The purpose of this framework is to discuss the soundness and completeness of combination approaches. It is not as detailed as the DPLL($\mathcal{T}$) framework [29] since it is not meant to be an operational description of solvers. In contrast to DPLL($\mathcal{T}$) and for simplicity, our schema highlights the distinction between the Boolean reasoning and the theory reasoning. The DPLL($\mathcal{T}$) framework can be seen as a refinement of our schema. Using our schema, we show the soundness and completeness of the most common architectures, as well as the new combination approach.

Overview of the paper. The next section introduces notations and the basics of SMT-solvers. In Section 3 we present the abstract framework for describing SMT-solvers. Section 4 uses the framework introduced in Section 3 to discuss the soundness and completeness of various approaches to SMT solving. A practical algorithm using this approach is presented in Section 5. Two illustrating examples are discussed in Sections 6 and 7. Section 8 shortly presents our implementation of the method, the veriT solver.

2. Basics of SMT-solvers

2.1. Notations and definitions

A *first-order language* is a tuple $\mathcal{L} = \langle \mathcal{V}, \mathcal{F}, \mathcal{P} \rangle$ such that $\mathcal{V}$ is an enumerable set of variables, $\mathcal{F}$ and $\mathcal{P}$ are sets of function and predicate symbols, and are such that $\mathcal{F} \cap \mathcal{P} = \emptyset$. To every function and predicate symbol an *arity* assigned. Nullary predicates are *propositions*, and nullary functions are *constants*. The set of *terms* over the language $\mathcal{L}$ is defined in the usual way. An *atomic formula* is either $t = t'$ where t and t' are terms, or a predicate symbol applied to the right number of terms. *Formulas* are built from atomic formulas, Boolean connectors $(\neg, \wedge, \vee, \Rightarrow, \equiv)$, and quantifiers $(\forall, \exists)$. A formula without quantifiers is called *quantifier-free*. A formula with no free variables is *closed*. A *theory* is a set of closed formulas. Two theories are *disjoint* if no predicate symbol in $\mathcal{P}$ and no function symbol in $\mathcal{F}$ appears in both theories.

An *interpretation* $\mathcal{I}$ for a first-order language provides a domain D, a total function $\mathcal{I}[f]$ on D with appropriate arity to every function symbol f, a predicate $\mathcal{I}[p]$ on D with appropriate arity to every predicate symbol p, and an element $\mathcal{I}[x]$ to every variable x. By extension, an interpretation gives a value in D to every term, and a truth-value to every formula. A *model* for a formula (or a theory) is an interpretation that makes the formula (resp. every formula in the theory) true. A formula is *satisfiable* if it has a model, and it is *unsatisfiable* otherwise. A formula φ is $\mathcal{T}$-satisfiable if it is satisfiable in the theory $\mathcal{T}$, that is, if $\mathcal{T} \cup \{\varphi\}$ is satisfiable. A $\mathcal{T}$-model of φ is a model of $\mathcal{T} \cup \{\varphi\}$. A formula φ is $\mathcal{T}$-unsatisfiable if it has no $\mathcal{T}$-model.

A formula is ($\mathcal{T}$-)valid if its negation is ($\mathcal{T}$-)unsatisfiable. The formula φ is a ($\mathcal{T}$-)*logical consequence* of the formula ψ, noted $\psi \models_{(\mathcal{T})} \varphi$ if every ($\mathcal{T}$-)model of ψ is a ($\mathcal{T}$-)model of φ. The logical consequence is also defined for sets of formulas, understanding a set of formulas as the conjunction of its components. A formula φ is ($\mathcal{T}$)-equisatisfiable to the formula ψ if φ is a ($\mathcal{T}$)-logical consequence of ψ and ψ is also a ($\mathcal{T}$)-logical consequence of φ.

An *atom* is an atomic formula. A *literal* is an atom or the negation of an atom. If ℓ is a literal, we implicitly consider that $\neg\neg\ell$ is the literal ℓ. A *conjunctive normal form* is a conjunction of *clauses*, i.e. a conjunction of disjunctions of literals. It is

[1] Intuitively, a theory is said to be convex whenever if a finite set of literals implies a disjunction of equalities then it implies at least one such equality. Linear integer arithmetic is not convex since $y \neq z \wedge 0 \leq x, y, z \leq 1$ implies that $x = y \vee x = z$, but implies neither $x = y$ nor $y = z$.

always possible to transform a quantifier-free formula into an equivalent or equisatisfiable conjunctive normal form [3,31]. We assume that clauses never contain twice the same atom; if a clause contains a literal and its negation, it reduces to the valid clause; redundant literals in clauses can be eliminated.

A theory $\mathcal{T}$ is said *convex*, if whenever a disjunction of equalities $\bigvee_{i=1}^{n} x_i = y_i$ is a logical consequence of a set of literals Γ (i.e. $\Gamma \models_{\mathcal{T}} \bigvee_{i=1}^{n} x_i = y_i$), then $\Gamma \models_{\mathcal{T}} x_i = y_i$ for some $i \in [1..n]$. A theory $\mathcal{T}$ is *stably infinite* if every $\mathcal{T}$-satisfiable set of literals Γ has an infinite $\mathcal{T}$-model.

A *propositional abstraction* for a set of formulas is this set of formulas in which every atom has been replaced by a proposition, all occurrences of the same atom being replaced by the same proposition. A set of clauses is *propositionally satisfiable* if its propositional abstraction is satisfiable. A *propositional assignment* Γ is a set of literals such that $\ell \notin \Gamma$ or $\neg\ell \notin \Gamma$ for every literal ℓ. By construction, a propositional assignment is a propositionally satisfiable set. A propositional assignment Γ is *total* with respect to a set of clauses if $\ell \in \Gamma$ or $\neg\ell \in \Gamma$ for every atom ℓ used in the set. A set of formulas G is a *propositional consequence* of a set of formulas H, if the propositional abstraction of G is a logical consequence of the propositional abstraction of H, the mapping from atoms to propositions being the same in both abstractions. An *entailing assignment* Γ for a set of clauses S is a propositional assignment such that S is a propositional consequence of Γ.

2.2. General overview

SMT-solvers are built by extending a propositional satisfiability solver, or SAT-solver for short, with decision procedures for the theories that appear in the formula. The SAT-solver is given the propositional abstraction of the input formula φ. For instance, consider φ is the following formula:

$$\neg\left(\left(\overbrace{x \le y}^{p_1} \wedge \overbrace{y \le x + f(x)}^{p_2} \wedge \overbrace{f(x) = 0}^{p_3} \wedge \overbrace{P(h(x) - h(y))}^{p_4}\right) \Rightarrow \left(\overbrace{P(0)}^{p_5} \wedge \overbrace{f(y) = 0}^{p_6}\right)\right).$$

Each atom of φ is abstracted to a propositional variable p_i. In practice, modern SAT-solvers [26,16] represent formulas in conjunctive normal form and are based on the Davis Logemann Loveland procedure [12]; for instance, the propositional abstraction of the previous formula would be represented as:

$$p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge (\neg p_5 \vee \neg p_6)\,.$$

If the formula φ is propositionally unsatisfiable, then it is also $\mathcal{T}$-unsatisfiable. Otherwise, the SAT-solver finds an entailing assignment Γ of the formula abstraction by searching an assignment of the propositional variables that satisfies each clause in the current set. For the above example, $\{p_1, p_2, p_3, p_4, \neg p_5\}$ is such an entailing assignment.

The $\mathcal{T}$-satisfiability of the conjunction of the literals corresponding to the abstracted literals needs then to be verified. For instance, the set of literals corresponding to the entailing assignment $\{p_1, p_2, p_3, p_4, \neg p_5\}$ is

$$\{x \le y, y \le x + f(x), f(x) = 0, P(h(x) - h(y)), \neg P(0)\}. \tag{2}$$

In general, the literals may contain symbols from several theories $\mathcal{T}_1, \mathcal{T}_2, \ldots \mathcal{T}_k$. In our example, literals mix uninterpreted symbols (f, h, P) with arithmetic symbols. An SMT-solver therefore needs a mechanism to combine the decision procedures for each $\mathcal{T}_i$ into a decision procedure for the composition of these theories, such as Shostak's [35] or Nelson and Oppen's [27]. In the following, we use the term *theory reasoner* to name such mechanisms.

In the Nelson and Oppen framework, the set of literals is used to produce a $\mathcal{T}$-equisatisfiable set $\Gamma' = \Gamma_1 \cup \Gamma_2 \cup \cdots \Gamma_k$ of *pure* literals, i.e. literals of each Γ_i only contain symbols from the theory $\mathcal{T}_i$. Such a separation is easily built by introducing new variables. For instance, a separation for the above set of literals (2) corresponding to the former entailing assignment $\{p_1, p_2, p_3, p_4, \neg p_5\}$ can be:

$$\Gamma_1 = \{x \le y,\ y \le x + v_1,\ v_1 = 0,\ v_2 = v_3 - v_4,\ v_5 = 0\}$$
$$\Gamma_2 = \{P(v_2),\ \neg P(v_5),\ v_1 = f(x),\ v_3 = h(x),\ v_4 = h(y)\}$$

where v_1 to v_5 are new variables, $\mathcal{T}_1$ and $\mathcal{T}_2$ are respectively the theories for linear arithmetic on real numbers and for uninterpreted functions. The set of shared variables is $\{v_1, v_2, v_3, v_4, v_5, x, y\}$. The decision procedures for each theory $\mathcal{T}_i$ receive the corresponding set of now pure literals Γ_i. Each decision procedure is thus able to decide if its set of (pure) literals is $\mathcal{T}_i$-satisfiable or not. The union of all pure sets in the separation being $\mathcal{T}$-equisatisfiable to the original set of literals, if one pure set in the separation is $\mathcal{T}_i$-unsatisfiable, then the original set is also $\mathcal{T}$-unsatisfiable. The converse is not true though: a $\mathcal{T}$-unsatisfiable set Γ of literals may produce a separation such that, for each i, the (pure) set Γ_i is $\mathcal{T}_i$-satisfiable. This is indeed the case in our example, since Γ_1 and Γ_2 are respectively $\mathcal{T}_1$-satisfiable and $\mathcal{T}_2$-satisfiable.

In fact, the combination of the decision procedures requires that each decision procedure propagates to the others all the deducible disjunctions of equalities between shared variables. If the signatures are disjoint and the theories stably infinite (as it is the case in our example), then the original set of literals is $\mathcal{T}$-unsatisfiable if and only if $\mathcal{T}_i$-unsatisfiability is derived by one of the decision procedures. In our example, the decision procedure for arithmetic can derive $x = y$. The equality $v_3 = v_4$ can be then deduced by the decision procedure for uninterpreted symbols from Γ_1 and $x = y$. The decision procedure for arithmetic can now derive the equality $v_2 = v_5$ which, in conjunction with Γ_2, can be determined to be $\mathcal{T}_2$-unsatisfiable by the decision procedure for uninterpreted symbols. Both decision procedures cooperated successfully to deduce the unsatisfiability of the original set Γ.

If, as in our example, the original set of literals Γ is $\mathcal{T}$-unsatisfiable, the propositional solver needs to be updated to prevent it from generating the same assignment again. From a logical viewpoint this corresponds to adding conjunctively the negation of the assignment. It is easy to see that this can be achieved by adding a new clause in the propositional SAT-solver, consisting of the negation of the literals in the assignment. For our example, the clause $\neg p_1 \vee \neg p_2 \vee \neg p_3 \vee \neg p_4 \vee p_5$ would be added. Such a clause, resulting from the negation of a $\mathcal{T}$-unsatisfiable assignment, is called a *conflict clause*.

If the original set of literals Γ is $\mathcal{T}$-satisfiable, then it is a $\mathcal{T}$-model of the original formula φ, which is itself $\mathcal{T}$-satisfiable. To understand this, notice that any $\mathcal{T}$-model of Γ translates to a $\mathcal{T}$-model of the conjunctive normal form of φ and is thus a $\mathcal{T}$-model of φ.

2.3. Practical considerations

In summary, SMT-solvers consist in the interplay between a SAT-solver and a theory reasoner, as schematized in Algorithm 1. The SAT-solver is responsible for maintaining a set of clauses, initially corresponding to a propositional abstraction of the formula, and producing candidate $\mathcal{T}$-models, such assignments being models at the propositional level (line 2). The theory reasoner is responsible for deciding if these assignments are indeed models or not (line 4), and the result of this decision is taken into account by the SAT-solver through the addition of conflict clauses (line 8). Several techniques have been proposed to increase the practical efficiency of this approach. These techniques consist in reducing the size of the search space of the SAT-solver either by increasing the number of clauses or by guiding the search into relevant regions. Generally, such techniques impose additional requirements on the theory reasoner or the decision procedures.

```
   Data: φ: formula
   Result: satisfiability status: SAT with assignment, or UNSAT
 1 SAT_solver.initialize(φ);
 2 while SAT_solver.status() = SAT do
 3 |   assignment := SAT_solver.assignment();
 4 |   theory_reasoner.set_literals(assignment);
 5 |   if theory_reasoner.status() = SAT then
 6 |   |   return SAT, assignment;
 7 |   end
 8 |   SAT_solver.add(theory_reasoner.conflict_clause);
 9 end
10 return UNSAT;
```

Algorithm 1: A basic SMT-solver.

First, when an assignment is found to be $\mathcal{T}$-unsatisfiable, instead of building the conflict clause from the whole assignment, it is sound to consider any $\mathcal{T}$-unsatisfiable subset of the assignment. Indeed, a smaller conflict clause results in pruning a larger number of propositional models from the search space of the SAT-solver.

In some cases, the decision procedures, or the combination framework, may be able to generate lemmas from φ that may be useful to guide the search and restrict the search space of the propositional SAT-solver. For instance, assuming t is a term with an integer type, and that the formula φ contains the atoms $0 < t$ and $t < 3$, adding the clause $\neg 0 < t \vee \neg t < 3 \vee t = 1 \vee t = 2$ introduces indirectly an integer arithmetic property at the propositional level.

Also for efficiency reasons, modern SMT-solvers use theory propagation [29], a technique that allows the theory reasoners to interact tightly with the SAT-solver engine. Internally, the SAT-solver works by extending a (initially empty) partial assignment towards a total assignment either by deducing propositional consequences of the partial assignment (this is called Boolean constraint propagation), or by deciding heuristically the value of an unassigned atom. Decisions may be backtracked when the assignment contradicts a clause, i.e. when a conflict is found. It may thus be valuable to avoid such heuristic decisions if it is possible. In the context of SMT-solvers, this is the role of theory propagation. For instance, assume that the atoms $x = y$ and $y = z$ are assigned to true by the SAT-solver, and that $f(x) = f(z)$ is an unassigned atom. Rather than letting the SAT-solver heuristically decide the value of an arbitrary atom, it may be in practice better to assign true to the atom $f(x) = f(z)$, since it is a $\mathcal{T}$-logical consequence of the assignment. The theory reasoner decreases the number of heuristic guesses and, consequently, restricts the search space of the propositional SAT-solver.

3. Soundness and completeness of SMT-solvers

Initially, the formula given as input to the SMT-solver is converted to a conjunctive set of clauses S (see for instance [3,31]). The goal of the SMT-solver is to decide whether S is $\mathcal{T}$-unsatisfiable or that S satisfiable. In the former case, the SMT-solver is said to be in the final state UNSAT. In the latter case, the SMT-solver shall additionally identify a boolean assignment Γ of the atoms occurring in S, such that S is $\mathcal{T}$-satisfiable. This corresponds to the final state denoted SAT(Γ). A sound SMT-solver will never get to the UNSAT final state on a $\mathcal{T}$-satisfiable formula, or decide satisfiability of a $\mathcal{T}$-unsatisfiable formula. It is complete if it always terminates.

$$\textsc{Bool}: \quad \frac{\langle S, \Gamma\rangle}{\langle S, \Gamma'\rangle} \qquad \Gamma' \text{ is a propositional assignment of } S \tag{3}$$

$$\textsc{Unsat}: \quad \frac{\langle S, \Gamma\rangle}{\text{UNSAT}} \qquad S \text{ is propositionally unsatisfiable} \tag{4}$$

$$\textsc{Learn}: \quad \frac{\langle S, \Gamma\rangle}{\langle S \cup \{C\}, \Gamma\rangle} \qquad S \models_{\mathcal{T}} C \tag{5}$$

$$\textsc{Sat}: \quad \frac{\langle S, \Gamma\rangle}{\text{SAT}(\Gamma)} \qquad \Gamma \text{ is entailing and } \mathcal{T}\text{-satisfiable} \tag{6}$$

Fig. 1. Rules representing the execution of an SMT-solver

The SAT-solver maintains a boolean assignment Γ of the atoms in the set of clauses. The pair $\langle S, \Gamma\rangle$ thus represents the current intermediate state of the solver. The set of rules given in Fig. 1, given in a SOS-like style, provides an abstract non-deterministic model of the possible behavior of the SMT-solver and is described in detail in the following. Observe that clauses may be added to the set S either by the SAT-solver itself, or by the theory reasoner (based on the propositional assignment from the SAT-solver). The reasoning ends when the SAT-solver concludes that the set of clauses is unsatisfiable, or when the theory reasoner asserts that the propositional assignment Γ is also $\mathcal{T}$-satisfiable.

Rule Bool (3) formalizes the update of the propositional assignment by the SAT-solver; Γ' is a new assignment such that $\Gamma' \cup \{C\}$ is propositionally satisfiable for every clause $C \in S$. The assignment is not required to be total; an assumption about assignment totality will be made later. The SAT-solver can also conclude that S is unsatisfiable using rule Unsat (4).

The addition of new clauses is represented by rule Learn (5). The new clause C may be added by the SAT-solver itself. In that case, C is a propositional consequence of S. The clause may also be added by the theory reasoner; C should then be a $\mathcal{T}$-logical consequence of the set S, according to the considered theory ($S \models_{\mathcal{T}} C$). By induction, it is clear that every added clause is a consequence of the original formula, and that the set of clauses is always $\mathcal{T}$-equisatisfiable to the original set of clauses.

When the assignment produced by the SAT-solver is entailing and $\mathcal{T}$-satisfiable, then the theory solver may conclude that the formula is $\mathcal{T}$-satisfiable. This is summarized in rule Sat (6).

In the present scheme, no assumption is made on the order of application of rules, on how the clause C is generated in Learn (5) and on the relation between consecutive assignments from the SAT-solver. This is all left abstract, with side conditions for the soundness and completeness of the SMT-solver.

Theorem 1. *An SMT-solver implementing the rules of Fig. 1 is sound.*

Proof. Initially, the set of clauses is just a conjunctive normal form of the input formula. We first prove by induction that the set of clauses given to the SAT-solver is always $\mathcal{T}$-equisatisfiable to the input formula. Assume that S and S' are the sets of clauses respectively before and after the application of a rule of Fig. 1. The sets S and S' only differ when rule Learn (5) is applied. In that case $S' = S \cup \{C\}$, with $S \models_{\mathcal{T}} C$. Thus S' is a $\mathcal{T}$-logical consequence of S; conversely, since S' contains S, S is also a $\mathcal{T}$-logical consequence of S'. In other words S and S' are $\mathcal{T}$-logically equivalent. By induction, the set of clauses is always $\mathcal{T}$-logically equivalent to the original set of clauses, and thus $\mathcal{T}$-equisatisfiable to the input formula.

If the SAT-solver concludes that the set of clauses is propositionally unsatisfiable (using rule Unsat (4)), the initial set of clauses and the input formula are unsatisfiable.

If rule Sat (6) is applied, then there exists a $\mathcal{T}$-satisfiable entailing assignment Γ of the set of clauses S. Assume $\mathcal{M}$ is a $\mathcal{T}$-model of Γ. Since Γ is a propositional model of S, $\mathcal{M}$ is a model of every clause in S. The set of clauses S, like the original formula, is thus satisfiable. □

Notice that the assumption in rule Learn (5) is very permissive. It holds notably for propositional learning, a technique used inside SAT-solvers, where the new clause C is obtained by propositional resolution of clauses in S, guided by a conflict analysis procedure known as the FUIP (First Unique Implication Point) computation [38]. It also holds for conflict clauses from the theory reasoner where the clause C is the disjunction of the negation of literals in a $\mathcal{T}$-unsatisfiable subset of the assignment Γ ($\mathcal{T}$ being the considered theory). Some further assumptions are however required to prove the completeness of an SMT-solver implementing the rules of Fig. 1.

Theorem 2. *An SMT-solver implementing the rules of Fig. 1 is complete (eventually terminates on a SAT(Γ) or UNSAT state) provided that*

- *on any set of clauses, the SAT-solver will eventually either*
  - *provide an entailing assignment*
  - *or conclude the unsatisfiability of the set of clauses with rule* Unsat (4)*;*

- *the atoms of the clauses added by rule* Learn (5) *belong to a finite set that is fixed* a priori *for the whole execution of the SMT-solver;*
- *for any state* $\langle S, \Gamma \rangle$ *where* Γ *is entailing, either rule* Sat (6) *is applied or rule* Learn (5) *is applied, with* C *not being a propositional consequence of* Γ.

Proof. If the run is finite then it should end either with rule Sat (6) or rule Unsat (4). This is proved by contradiction. Assume the run is finite but does not terminates on an UNSAT or SAT state. Then the ending state is of the form $\langle S, \Gamma \rangle$. The first assumption implies that Γ is entailing. Since Γ is entailing, the last assumption ensures either that

- the new state is SAT (with the application of rule Sat (6)) or
- the rule Learn (5) is applied and introduces a clause C that is not a propositional consequence of Γ.

The first option is not possible, since it contradicts the hypothesis that the ending state is not a SAT state. The second option is also not possible, since this would change the set S, and contradicts the fact that $\langle S, \Gamma \rangle$ is the ending state.

Assume now that the run is infinite. The set of atoms that is or will be present in the set of clauses is finite, thanks to the second assumption. The set of possible different clauses is also finite. At some point no new clause will be added to the set of clauses S by rule Learn (5), and the SAT-solver will eventually provide an entailing assignment Γ. The rule Sat (6) being an ending rule, the next rule will be rule Learn (5), and a clause C will be generated. Since C already belongs to the set of clauses and Γ is entailing, then C is a logical consequence of Γ which contradicts the last assumption of the theorem. □

The three requirements in the above theorem are reasonable. The first requirement is on the SAT-solver: on any set of clauses, it should decide that it is unsatisfiable, or provide a total (and thus entailing) assignment. This requirement is fulfilled by existing tools [26,16]. The two remaining requirements are related to the theory reasoner and are discussed in the next section.

4. Combination of theories and propositional reasoning

The previous framework can easily be instantiated to common approaches of combinations of theories. We here discuss these approaches, beginning with the original combination schema [27,36], and a practical implementation based on arrangement guessing by the SAT-solver called Delayed Theory Combination [10]. In Section 4.2, we review another presentation of the original schema [27] based on equality propagation, and an implementation refinement referred as Splitting on Demand [5]. Finally we introduce the new schema that propagates equalities that are either deduced or generated by generalizing models.

4.1. Nelson–Oppen and arrangements

As discussed in Section 2.2, to study the $\mathcal{T}$-satisfiability of a set of literals Γ, where the theory $\mathcal{T}$ is the union of the two disjoint stably infinite[2] theories $\mathcal{T}_1$ and $\mathcal{T}_2$, one traditionally first build a separation (Γ_1, Γ_2) such that $\Gamma_1 \cup \Gamma_2$ is $\mathcal{T}$-equisatisfiable to Γ, and Γ_1 only contains interpreted symbols from $\mathcal{T}_1$ (similarly for Γ_2). The separation ensures that the only shared symbols between Γ_1 and Γ_2 are variables and the equality predicate. Those shared variables play a special role in the combination. An arrangement $\mathcal{A}$ of a set of shared variables is a set that contains either an equality $X = Y$ or an inequality $X \neq Y$ for every pair of variables X, Y in the set.

Assume for instance we want to study the $\mathcal{T}$-satisfiability of the separation

$$\begin{aligned} \Gamma_1 &= \{x \leq 2,\ 0 < x,\ v_1 = 1,\ v_2 = 2\} \\ \Gamma_2 &= \{f(x) \neq f(v_1), f(x) \neq f(v_2)\} \end{aligned} \tag{7}$$

where $\mathcal{T}$ is the union of the theory of linear arithmetic on integer and the theory of uninterpreted symbols. The shared variables are $\{x, v_1, v_2\}$. There exist five arrangements for those variables: $\mathcal{A}_1 = \{x \neq v_1, x \neq v_2, v_1 \neq v_2\}$, $\mathcal{A}_2 = \{x = v_1, x \neq v_2\}$, $\mathcal{A}_3 = \{x \neq v_1, x = v_2\}$, $\mathcal{A}_4 = \{x \neq v_1, v_1 = v_2\}$, $\mathcal{A}_5 = \{x = v_1, x = v_2\}$ (redundant (in)equalities have been ignored).

The Nelson–Oppen combination scheme (see for instance [27,36]) is based on the following theorem. This result reduces the $\mathcal{T}$-satisfiability problem of a set of literals to $\mathcal{T}_1$-satisfiability and $\mathcal{T}_2$-satisfiability problems:

Theorem 3. *Let* $\mathcal{T}$ *be the union of the disjoint and stably infinite theories* $\mathcal{T}_1$ *and* $\mathcal{T}_2$. *A separation* $\Gamma_1 \cup \Gamma_2$ *— where* Γ_1 *only contains symbols from* $\mathcal{T}_1$, *the equality symbol and variables; similarly for* Γ_2 *— is* $\mathcal{T}$*-satisfiable if and only if there exists an arrangement* $\mathcal{A}$ *of the shared variables between* Γ_1 *and* Γ_2 *such that* $\mathcal{A} \cup \Gamma_1$ *is* $\mathcal{T}_1$*-satisfiable and* $\mathcal{A} \cup \Gamma_2$ *is* $\mathcal{T}_2$*-satisfiable.*

Proof. Let V be the set of shared variables between Γ_1 and Γ_2.

The condition is necessary. Let $\mathcal{I}$ be a $\mathcal{T}$-model of $\Gamma_1 \cup \Gamma_2$. This interpretation $\mathcal{I}$ perfectly defines an arrangement $\mathcal{A}$ of V, and $\mathcal{I}$ is a model of $\mathcal{A} \cup L_1 \cup L_2$, and so of $\mathcal{A} \cup L_i$ for $i = 1, 2$.

To prove that the condition is sufficient, assume that $\mathcal{A} \cup \Gamma_1$ is $\mathcal{T}_1$-satisfiable and $\mathcal{A} \cup \Gamma_2$ is $\mathcal{T}_2$-satisfiable. Since $\mathcal{T}_i$ is a first-order stably infinite theory for $i = 1, 2$, there exists an interpretation $\mathcal{I}_i$ on a domain D_i of cardinality $\aleph_0$ that makes

[2] Disjointness and stably infiniteness are sufficient conditions for the easy combination of two theories. These conditions are not necessary (see for instance [17,37]), but for simplicity, we assume those conditions are met.

true $\mathcal{A} \cup L_i$. Since both interpretations are models of $\mathcal{A}$ and since both domains have the same cardinality, it is possible to build a bijection b from D_2 to D_1 such that $b(\mathcal{I}_2[x]) = \mathcal{I}_1[x]$ for every shared variable $x \in V$. The interpretation $\mathcal{I}$ is defined on domain D_1. For every function symbol f in $\mathcal{T}_1$, we define $\mathcal{I}[f] = \mathcal{I}_1[f]$, and similarly for predicate symbols in $\mathcal{T}_1$ and shared variables in V. For every function symbol f of arity n in $\mathcal{T}_2$, we define $\mathcal{I}[f]$ such that, for any $d_1, \ldots d_n \in D_1$, $\mathcal{I}[f](d_1, \ldots d_n) = b(\mathcal{I}_2[f](b^{-1}(d_1), \ldots b^{-1}(d_n)))$, and similarly for predicate symbols. $\mathcal{I}$ is a $\mathcal{T}$-model of $\mathcal{A} \cup \Gamma_1 \cup \Gamma_2$, and thus of $\Gamma_1 \cup \Gamma_2$. □

For the example (7), $\mathcal{A}_1 \cup \Gamma_1$, $\mathcal{A}_4 \cup \Gamma_1$ and $\mathcal{A}_5 \cup \Gamma_1$ are unsatisfiable in the theory of linear arithmetic on integer. For the remaining arrangements, $\mathcal{A}_2 \cup \Gamma_2$ and $\mathcal{A}_3 \cup \Gamma_2$ are unsatisfiable according to the theory of uninterpreted symbols. Every arrangement leads to unsatisfiability in one or the other theory (or both), and indeed, $\Gamma_1 \cup \Gamma_2$ is unsatisfiable in the union of theories.

The same approach can be used for the example in Section 2; for instance, an arrangement for the set of variables $\{v_1, v_2, v_3, v_4, v_5, x, y\}$ could be $\{v_1 = v_2, v_1 \neq v_3, v_3 = v_4, v_4 = v_5, x \neq v_1, x \neq v_3, x = y\}$ (redundant (in)equalities have been ignored). However enumerating all arrangements here already becomes difficult. For seven variables, in our toy example, there are already 877 different arrangements. The number of arrangements (i.e. the number of partitions, known as the Bell numbers) grows faster than exponentially with respect to the size of the set of variables. The naive approach — that is, extensively testing all arrangements — is thus only tractable for very small problems.

However the approach called *Delayed Theory Combination* (see [10]) is a successful and simple technique that delegates the job of enumerating arrangements to the SAT-solver. The formula is purified[3] before it is given to the SAT-solver, and the (entailing) total assignments from the SAT-solver should give a truth-value to every equality between shared variables. The SAT-solver implements rule Bool (3) when it generates an assignment and rule Unsat (4) when it cannot build such an assignment. The assignment is then given to every decision procedure in the theory reasoner; each decision procedure only picks among the assignment the set of literals that is relevant to the corresponding theory. This technique is implemented in several state of the art SMT-solvers. When given a total assignment Γ, one decision procedure can conclude that its set is unsatisfiable and then it returns a conflict clause of the form $\bigvee_{\ell \in \gamma} \neg \ell$ where γ is an unsatisfiable subset of Γ; this clause is obviously not a logical consequence of Γ and is used to implement rule Learn (5). In the other case, every decision procedure concludes that its set is satisfiable, and Γ is also satisfiable in the combination of theories since every (entailing) total assignment contains an arrangement. Rule Sat (6) is thus instantiated. According to Section 3, the approach is thus sound and complete.

4.2. Nelson–Oppen and deduced disjunctions of equalities

Another practical way is to build the arrangement using disjunctions of equalities deduced from the independent theories. Instead of guessing the arrangement, the decision procedures impose constraints on arrangements that may satisfy all theories in the combination. A first example of this technique has been given in Section 2.2, where the unsatisfiability of the separation for the set of literals (2) is deduced by the exchange of deduced equalities between the decision procedure for linear arithmetic on real numbers and the one for uninterpreted symbols. The simplicity of this case is due to the convexity of the theories in the combination: if a disjunction of equalities can be deduced by a convex theory, there exists an equality in the disjunction that can also be deduced.

Not all theories are convex, and for instance linear arithmetic on integers is not. In example (7) no equality can be deduced from either Γ_1 or Γ_2. However, $x = v_1 \vee x = v_2$ is a logical consequence of Γ_1 according to linear arithmetic on integers. Splitting on the equalities in this disjunction quickly shows that the $\Gamma_1 \cup \Gamma_2$ is unsatisfiable in the combination of the theories.

The completeness of the approach using deduced disjunctions of equalities is guaranteed by the following theorem. Informally, it states that if a set of literals is satisfiable in two disjoint (stably-infinite) theories $\mathcal{T}_1$ and $\mathcal{T}_2$, and no disjunction of equalities can be deduced from either $\mathcal{T}_1$ or $\mathcal{T}_2$, then it is satisfiable in $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$. Consequently, completeness of a cooperation of decision procedures can be obtained by exhaustively exchanging disjunctions of equalities between the decision procedures (instead of checking all arrangements).

Theorem 4. *Assume Γ_1 is $\mathcal{T}_1$-satisfiable, Γ_2 is $\mathcal{T}_2$-satisfiable, but $\Gamma_1 \cup \Gamma_2$ is not $\mathcal{T}$-satisfiable. Then there exists a disjunction of equalities $\Delta = \bigvee_{i=1}^{n} x_i = y_i$ (where x_i and y_i are shared variables) such that*

- *$\Gamma_1 \models_{\mathcal{T}_1} \Delta$ and $\Gamma_2 \not\models_{\mathcal{T}_2} x_i = y_i$ for each i such that $1 \leq i \leq n$;*
- *or $\Gamma_2 \models_{\mathcal{T}_2} \Delta$ and $\Gamma_1 \not\models_{\mathcal{T}_1} x_i = y_i$ for each i such that $1 \leq i \leq n$.*

Proof. Let S_1 be the set of all equalities $x = y$ such that $\Gamma_1 \models_{\mathcal{T}_1} x = y$, and S_2 be the set of all equalities $x = y$ such that $\Gamma_2 \models_{\mathcal{T}_2} x = y$. Assume that $x_1 = y_1$ belongs to S_1 but not to S_2: then the required new deduced disjunction of equalities can simply be this single equality (and similarly for an equality that belong to S_2 but not S_1). It remains to consider $S_1 = S_2$. The set S_1 (or equivalently S_2) is extended to form an arrangement $\mathcal{A}$ by adding $x \neq y$ to S_1 whenever $x = y$ does not belong to S_1. According to Theorem 3, $\mathcal{A} \cup \Gamma_1$ is $\mathcal{T}_1$-unsatisfiable or $\mathcal{A} \cup \Gamma_2$ is $\mathcal{T}_2$-unsatisfiable since $\Gamma_1 \cup \Gamma_2$ is $\mathcal{T}$-unsatisfiable. Assume that

[3] Purification ensures every atom contains interpreted symbols from only one theory. It has the same effect than building a separation, but it is done once and for all in the original formula and not on successive assignments.

$\mathcal{A} \cup \Gamma_1$ is $\mathcal{T}_1$-unsatisfiable (the other case is handled similarly). Then $\Gamma_1 \models_{\mathcal{T}_1} \bigvee_{x \neq y \in \mathcal{A}} x = y$ and by construction $\Gamma_2 \not\models x = y$ if $x \neq y$ belongs to $\mathcal{A}$. □

The application of the previous result requires splitting at the theory level. Assume $\Gamma_1 \models_{\mathcal{T}_1} \bigvee_{i=1}^{n} x_i = y_i$. Then, for every i, $\Gamma_2 \cup \{x_i = y_i\}$ is checked for satisfiability. If there exists i such that $\Gamma_2 \cup \{x_i = y_i\}$ is satisfiable, there may also exists another disjunction of equalities $\bigvee_{i=1}^{n'} x'_i = y'_i$ such that $\Gamma_2 \cup \{x_i = y_i\} \models \bigvee_{i=1}^{n'} x'_i = y'_i$. This new disjunction would also imply some splitting. Eventually no more disjunction of equalities will be generated and both theories in the combination will conclude that their respective set of literals (plus the equalities coming from the case splittings) is satisfiable, or a conflict will occur and it will be required to backtrack on the case splits to examine every choice.

If case splitting and backtracking is realized inside the theory reasoner for the combination of theories, the theory reasoner is complete by itself. When given an entailing assignment Γ from the SAT-solver, it will either conclude to the $\mathcal{T}$-satisfiability of the assignment ($\mathcal{T}$ being the considered combination of theories), or it returns a conflict clause of the form $\bigvee_{\ell \in \gamma} \neg \ell$ where γ is an unsatisfiable subset of Γ; once again, this clause is obviously not a logical consequence of Γ. No new atom is generated by the process, and all conditions are met for Theorems 1 and 2 to be applicable. The approach is sound and complete.

Some theories are particularly appropriate for the above method. For instance, no splitting is required when only combining convex theories. Many useful theories are convex, and notably linear arithmetic on the real numbers, and the theory of uninterpreted functions. Among the non-convex theories one finds the theory of linear arithmetic on the integers, and the theory of arrays.

In presence of non-convex theories, handling case splittings at the theory level may be difficult, and also inefficient. Usually this work is preferably delegated to the SAT-solver. This technique is referred as *Splitting on Demand* [5]. In such a case, rather than splitting on a disjunction $\bigvee_{i=1}^{n} x_i = y_i$ that is a $\mathcal{T}_i$-logical consequence of the set of literals Γ_i, a new clause $\bigvee_{j=1}^{n} x_j = y_j \vee \bigvee_{\ell \in \Gamma_i} \neg \ell$ is added to the SAT-solver. Most preferably, only the literals $\ell \in \Gamma_i$ that are required to imply the disjunction of equalities are included in the clause, so that the added clause subsumes many other clauses that would be generated in similar cases.

In contrast to Delayed Theory Combination and splitting inside the theory reasoner, handling case splitting through the SAT-solver may introduce new atoms, namely new equalities between terms of the formula. However, since there is only a finite number of terms in the original formula, and thus a finite number of possible equalities between terms from the original formula, the second assumption of Theorem 2 is fulfilled. For the last assumption to be fulfilled, it is sufficient to require from the theory reasoner that, if it is not able to state if the assignment is $\mathcal{T}$-satisfiable or not, it should at least provide a deduced clause. According to Theorem 4 such a clause exists whenever the assignment is $\mathcal{T}$-unsatisfiable; this clause is not a propositional consequence of the assignment. If the decision procedures are complete with respect to the deduction of disjunction of equalities, the SMT-solver is sound and complete.

For some theories (and in particular, for linear arithmetic over the integer) it is not easy to be complete for the deduction of disjunction of equalities (see for instance [22]). Moreover some decision procedures for convex theories are not easily tweaked to produce equalities between variables. The next section presents another way to ensure the completeness of SMT-solvers in those cases.

4.3. Introducing model equalities

The method we present here and in the following sections is suitable for decision procedures that are not able to deduce disjunctions of equalities, or that are not complete with respect to deduction of (disjunctions of) equalities. We assume they are however able to find a concrete model for a set of constraints, i.e. literals. As an example, it means that a decision procedure for integer linear arithmetic is able to find a mapping from variables to integers such that all constraints are satisfiable. Many decision procedures inherently have such a capability.

Assume that an assignment Γ provided by the SAT-solver produces (pure) literals Γ_1 and Γ_2 to be handled by theory reasoners for $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. Also assume that Γ_1 is $\mathcal{T}_1$-satisfiable, and Γ_2 is $\mathcal{T}_2$-satisfiable. Finally assume that all generated disjunctions of equalities have been handled as in the previous subsection. The theory reasoners that are not complete with respect to deduction of (disjunctions of) equalities should then compute a model, and generate the equalities between shared variables that correspond to the model and that do not already belong to Γ. These equalities are then given to the other decision procedure, as if they were in the original assignment. Those equalities may themselves force the other decision procedure to deduce or produce other equalities. Eventually no more equality is shared. If a conflict occurs, the theory reasoner for $\mathcal{T}_i$ generates a conflict clause C of the form $\bigvee_{\ell \in \gamma} \neg \ell$ where γ is a $\mathcal{T}_i$-unsatisfiable subset of $\Gamma \cup \Gamma'$ with Γ' being the set of all generated equalities. This clause is added to the set of clauses handled by the SAT-solver. It may contain atoms (equalities) coming from models. If it does not, it is conflicting in the sense that $\Gamma \cup \{C\}$ is unsatisfiable. If it does, it is obviously not a propositional consequence of Γ. The atoms generated here once again all belong to a finite set that is known *a priori*, namely the set of all equalities between two terms in the original formula.

If no conflict occurs, then $\Gamma \cup \Gamma'$ contains equalities between any two shared variables that are equal according to the model. Conversely, if an equality between two shared variables does not belong to $\Gamma \cup \Gamma'$, it has not been guessed nor deduced by the decision procedures in the combination. If we assume every decision procedure is either complete with respect to the generation of disjunction of equalities, or that it generates model equalities, one can conclude that the two

theories agree that, if no equality between two shared variables exists in $\Gamma \cup \Gamma'$, they should be different. An arrangement $\mathcal{A}$ can thus be built from the equalities in $\Gamma \cup \Gamma'$, augmented by the maximum number of inequalities between shared variables. $\mathcal{A} \cup \Gamma_1$ is $\mathcal{T}_1$-satisfiable and $\mathcal{A} \cup \Gamma_2$ is $\mathcal{T}_2$-satisfiable. Rule Sat (6) can be applied and the third assumption of Theorem 2 is fulfilled. The approach is sound and complete. Sections 6 and 7 contain two examples illustrating this approach, using the algorithm presented in Section 5.

5. An algorithm for Nelson–Oppen with model-equalities

Algorithm 2 provides a high level pseudo-code of the Nelson and Oppen framework with model equalities, as described in Section 4.3. We assume that the propositional satisfiability solver can incorporate new literals on-the-fly (corresponding to constraints that are not in the original formula). This capability is also required for the splitting on demand approach described in Section 4.2. The presented algorithm also gives the decision procedures the opportunity to take advantage of the similarities between the consecutive sets of literals produced by the propositional SAT-solver as they may update their state to reflect only the difference between these sets. This is the case when there exists efficient decision procedures that incrementally stack new literals and backtrack while maintaining the $\mathcal{T}$-satisfiability status of the current set of literals.

The set of theories in the combination is denoted $S_{\mathcal{T}}$, and the subset of theories that is not convex or with a decision procedure that is not able to deduce all the implied equalities is denoted $S_{\mathcal{T},\text{mod}}$. As described in Section 4.3, it is assumed that the decision procedure for each theory in $S_{\mathcal{T},\text{mod}}$ is able to generate the *model equalities* from a model they maintain based on the literals (including equalities and inequalities) they receive. The main difference with respect to a version without model equalities is located in the lines 16 to 18. If $S_{\mathcal{T},\text{mod}}$ is empty the version without model equalities is complete. This is the case when the theories in the combination are all convex and stably infinite (notice that every convex first-order theory with no trivial models is stably infinite [7]), and their decision procedure is able to deduce all implied equalities between shared variables.

The main loop of the algorithm is executed until the SAT-solver can no longer produce a propositional satisfiable assignment (line 1). In this case, the original formula is unsatisfiable (rule Unsat (4)). Otherwise, a propositional model is computed (line 2) (rule Bool (3)). Each decision procedure t may then backtrack to a state based on the new set of literals corresponding to this assignment (line 4). Note that the set of literals available in such state should be $\mathcal{T}_i$-satisfiable in each theory $\mathcal{T}_i$. The variable *new_literals* represents the set of literals that the decision procedures need to receive. It is initially set with the new literals present in the assignment (line 5), and is later updated with equalities produced by the decision procedures (lines 15 and 17). This set is repeatedly propagated to each decision procedure (and a separation is built on-the-fly) through the *propagate*() function (line 8) until one of them detects unsatisfiability (line 9) or no new equalities can be deduced (line 19). If unsatisfiability is detected, then a conflict clause is generated and added to the propositional satisfiability solver (line 10). This action implements an instance of rule Learn (5). Otherwise, each decision procedure computes the set of variable equalities entailed by the current set of literals. These sets are stored (line 15) for propagation at the next iteration. Ultimately, if no variable equalities can be deduced, then the decision procedures that do not have complete (disjunctions of) equality deduction capabilities will look for *model equalities* to propagate. In this algorithm, we assume that none of the non-convex theories has the capabilities of generating disjunctions of equalities, i.e. an internal case split is not handled.

Once all the literals and equalities have been propagated, additional lemmas produced by the decision procedures may be incorporated as clauses to the propositional satisfiability solver (lines 20, 21). Again, this corresponds to an instance of rule Learn (5). Eventually, when no new information can be provided to the propositional satisfiability solver, and if the assignment is total, then the algorithm concludes that the original formula is $\mathcal{T}$-satisfiable and halts (line 23), which corresponds to rule Sat (6).

6. Combining with model equalities: an example

In this section, we present an example illustrating the cooperation using *model equalities* between the theory of uninterpreted functions (UF) and a fragment of linear arithmetic, i.e. integer difference logic (IDL); in our framework, the decision procedures are not required to generate all the implied disjunctions of equalities, that would be otherwise necessary for completeness in a classical Nelson and Oppen combination framework. Difference Logic is the linear arithmetic fragment that contains only constraints of the kind $x - y \bowtie c$, where x and y are variables, c is a constant number and $\bowtie \in \{\leq, \geq, =, <, >\}$. Assume we want to prove that the following formula is unsatisfiable:

$$x \leq y + 1 \wedge y \leq x \wedge x \neq y \wedge f(x) \neq f(y + 1). \tag{8}$$

As a first step and for simplicity of the presentation, we assume the formula is purified (i.e. the *separation* is done at the formula level) so that the different decision procedures only get literals with symbols from their theory.[4]

$$\overbrace{v_1 = y + 1}^{p_1} \wedge \overbrace{x \leq v_1}^{p_2} \wedge \overbrace{y \leq x}^{p_3} \wedge \overbrace{x \neq y}^{\neg p_4} \wedge \overbrace{f(x) \neq f(v_1)}^{\neg p_5}. \tag{9}$$

Every atom is assigned a propositional variable p_i.

[4] Another approach, used in veriT, it to build the separation on-the-fly at the theory reasoner level.

Data: φ: formula
Result: satisfiability status: SAT with assignment, or UNSAT

```
1  while SAT_solver.status() = SAT do
2      assignment := SAT_solver.assignment();
3      status := SAT;
4      foreach t ∈ S_T do t.back_jump(assignment);
5      new_literals := assignment.get_new_literals();
6      repeat
7          foreach t ∈ S_T do
8              status := t.propagate(new_literals);
9              if status = UNSAT then
10                 SAT_solver.add(t.conflict_clause);
11                 break;
12             end
13         end
14         if status = UNSAT then break;
15         new_literals := ⋃_{t∈S_T} t.get_new_equalities();
16         if new_literals = ∅ then
17             new_literals := ⋃_{t∈S_{T,mod}} t.get_new_model_equalities();
18         end
19     until new_literals = ∅ ;
20     lemmas = ⋃_{t∈S_T} t.get_new_lemmas();
21     SAT_solver.add(lemmas);
22     if lemmas = ∅ ∧ status = SAT ∧ assignment.is_total() then
23         return SAT, assignment;
24     end
25 end
26 return UNSAT;
```

Algorithm 2: A SMT-solver with model-equality generation.

Fig. 2 can be used to trace the status of the algorithm during its application to this problem. In Fig. 2 and in the following, the symbols p_i may be used to denote the constraints to which they correspond. Also $a \neq b$ denotes $\neg(a = b)$, and $a > b$ (or $a < b$) may be used instead of $\neg(a \leq b)$ (respectively $\neg(a \geq b)$). $\mathcal{T}_1$ and $\mathcal{T}_2$ are the theories for uninterpreted functions (UF) and integer difference logic (IDL) respectively.

At State 1.0, the SAT-solver has found an entailing assignment for the formula. In this assignment, unsurprisingly, p_1, p_2 and p_3 are assigned to true, whereas p_4 and p_5 are assigned to false. This assignment is propagated to the decision procedures to check for theory consistency. At this point, no equality can be generated. The process would stop here if there were only convex theories involved.

However, IDL is non-convex. To obtain completeness of the cooperation of the decision procedures, two approaches are possible. Either, as in the classical combination framework (Section 4.2) one can propagate an implied disjunction of equalities, or, using the new approach proposed in Section 4.3 and implemented in Algorithm 2, one can simply generate a model equality. It is easy to tweak a difference logic solver based on graph algorithms (see e.g. [23]) to maintain a model, i.e. to assign to every variable a concrete integer value so that all constraints are satisfied. In such a model for State 1.0 — the set of constraints given to the IDL decision procedure is $\{p_1, p_2, p_3, \neg p_4\}$ or equivalently $\{v_1 = y + 1, x \leq v_1, y \leq x, x \neq y\}$ — a suitable model would assign $x = 0$, $y = -1$ and $v_1 = 0$. The *model equality* $x = v_1$ (abstracted to a new proposition p_6) is generated and propagated to all decision procedures. The resulting state is State 1.1, and a contradiction is found in the UF decision procedure, since the conjunction $x = v_1 \wedge f(x) \neq f(v_1)$ is unsatisfiable.

The conflict clause $p_5 \vee \neg p_6$ is added to the SAT-solver. A second iteration of the main loop is necessary. The SAT-solver is asked for a new propositional assignment, the decision procedures backtrack to the previous satisfiable state, i.e. just before the model equality $x = v_1$ (i.e. p_6) was propagated. The set of literals for each decision procedure is updated to reflect the change in the SAT-solver assignment: the literal $\neg p_6$ (i.e. $x \neq v_1$) is added to both sets. No contradiction or equality is generated by either decision procedure, but the current assignment of concrete values to variables for IDL is not consistent with the current assignment from the SAT solver, in particular since it contradicts $\neg p_6$ (i.e. $x \neq v_1$). Our implementation of the IDL decision procedure is not able to automatically handle a negation of equalities; in order to repair the model, the IDL decision procedure generates a *lemma*: $(x = v_1) \vee (x > v_1) \vee (x < v_1)$ (or equivalently $p_6 \vee \neg p_2 \vee \neg p_7$, where p_7 is a new propositional variable corresponding to the atom $v_1 \leq x$). The lemma is given to the SAT-solver, which therefore refines the propositional assignment to include $\neg p_7$. At this state (State 3.0 on Fig. 2), the IDL decision procedure is able to deduce the *equality between shared variables* $x = y$ from $x < v_1$, $v_1 = y + 1$ and $y \leq x$. At State 3.1, this equality is propagated to the decision procedure for UF, which detects the conflict $x \neq y \wedge x = y$, so the assignment is once again considered theory inconsistent. The corresponding conflict clause is generated ($p_4 \vee p_7 \vee \neg p_1 \vee \neg p_3$) and added to the SAT-solver.

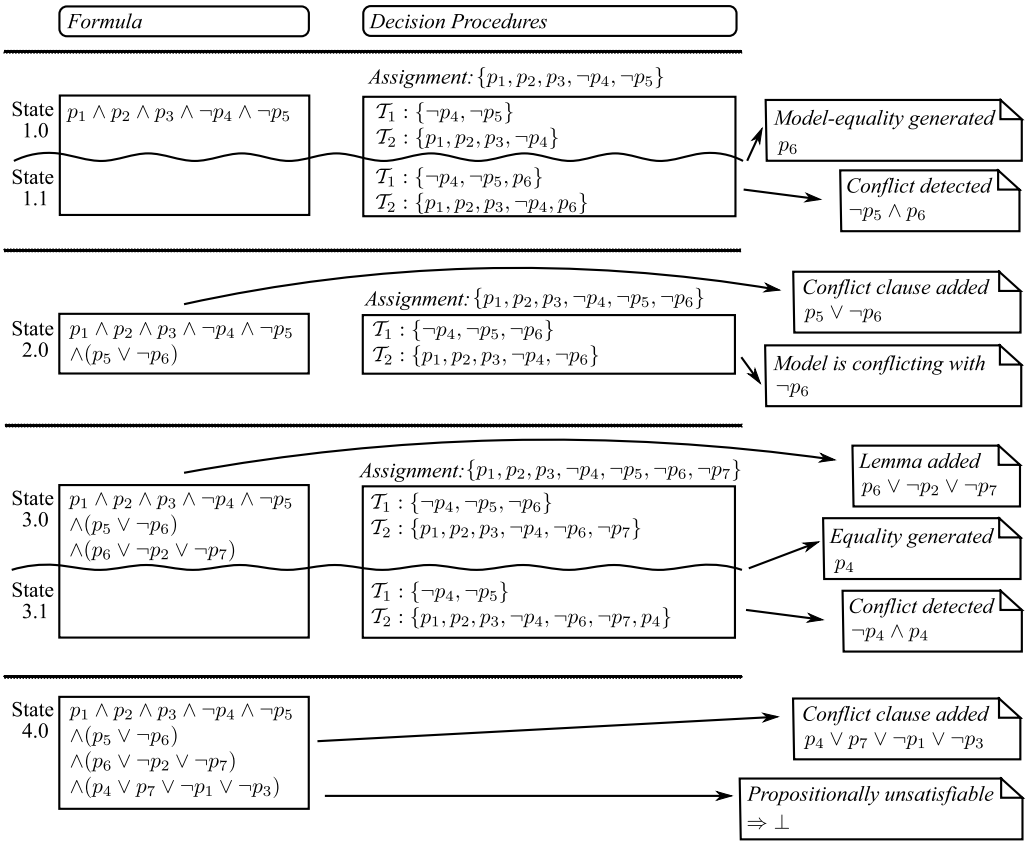

Fig. 2. An example combining UF and IDL.

Finally, at State 4, the SAT-solver concludes there is no more assignment to make the current formula propositionally true. Therefore, the problem is unsatisfiable.

7. A second example: combining uninterpreted functions with non-linear arithmetic

The previous example highlights the cooperation between the SAT-solver and the combination of theories. In this section, we show that the theory of uninterpreted functions (UF) and non-linear arithmetic (NLA) can cooperate by exchanging only equalities (some of them being *model-equalities*). The details of the interplay between the theory reasoner and the SAT-solver are abstracted to focus on the equality exchanges between the decision procedures. The internal details of the decision procedures are also quietly ignored. It is just assumed that it is possible to retrieve implied equalities from the constraint set and to detect unsatisfiability. It is also assumed that the non-linear arithmetic decision procedure can maintain a concrete model, assigning values to variables. This model will be helpful when generating model-equalities.

To study the satisfiability of the following formula

$$x^2 = 1 \wedge y^2 = 4 \wedge f(2x) = 1 \wedge f(y) = 0 \wedge f(-y) = 0,$$

as in the previous example, the formula is first purified:

$$x^2 = 1 \wedge y^2 = 4 \wedge f(v_1) = v_3 \wedge f(y) = v_4 \wedge f(v_2) = v_4 \wedge v_1 = 2x \wedge v_2 = -y \wedge v_3 = 1 \wedge v_4 = 0.$$

The constraints are first dispatched to their respective decision procedures for theory $\mathcal{T}_{NLA}$ (non-linear arithmetic) and theory $\mathcal{T}_{UF}$ (uninterpreted symbols). This is shown in Fig. 3, at State 1.0.

Each iteration proceeds as shown in Algorithm 2. At State 1.0, both decision procedures for $\mathcal{T}_{NLA}$ and $\mathcal{T}_{UF}$ conclude that their set of constraints is satisfiable. Furthermore, no equality between variables can be deduced.

However $\mathcal{T}_{NLA}$ is not convex. Again, it is not correct to conclude, in State 1.0, that the set of constraints is satisfiable. Either all disjunctions of equalities have to be produced and propagated, or, alternatively, one can produce *model-equalities*. At State 1.1, we assume that the decision procedure for $\mathcal{T}_{NLA}$ has generated the concrete model $v_2 = -2, v_4 = 0, x = v_3 = 1$ and $y = v_1 = 2$, and thus can indeed generate two *model-equalities* that are propagated to the decision procedure for $\mathcal{T}_{UF}$. At State 1.2, $\mathcal{T}_{UF}$ deduces the equality $v_3 = v_4$ that results in a conflict on the arithmetic side, once propagated. This conflict is translated to a clause and sent to the SAT-solver which will learn about the theory inconsistency and will generate a new assignment.

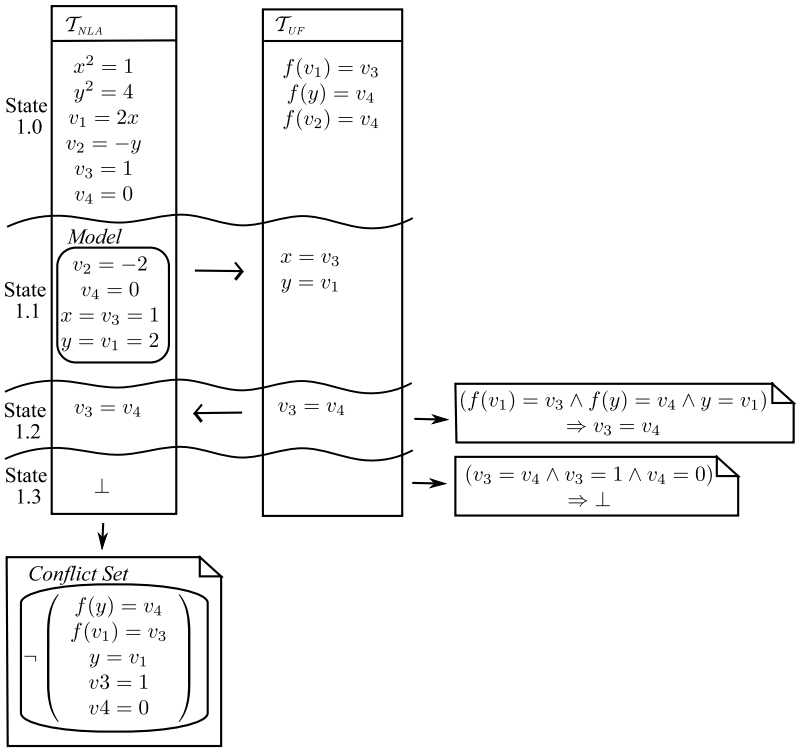

Fig. 3. Example UF and NLA: Trying the first assignment.

This new assignment will naturally assign false to the new constraint $y = v_1$, in order to make the new conflict clause true (Fig. 4, state 2.0). The iteration process is similar, but because of the new constraint, the model generated by the decision procedure for $\mathcal{T}_{NLA}$ is different from the previous one, which results in yet another *model-equality*. The final result is the same: after the deductions and propagations, $\mathcal{T}_{NLA}$ finds a conflict.

The third assignment (Fig. 5, State 3.0) differs from the previous one by one new constraint only. Now $\mathcal{T}_{NLA}$ finds an inconsistency directly, and no arithmetic model can be found anymore. Including this new conflict clause will result in a formula which is propositionally unsatisfiable. Therefore the original formula is unsatisfiable.

It is worth noticing that, by combination of all possible values of x and y (according to constraints $x^2 = 1$ and $y^2 = 4$), four different concrete arithmetic models are possible. But thanks to the learning process, just two of them need to be examined to conclude the unsatisfiability of the formula.

8. An implementation: the SMT-solver veriT

The veriT solver provides an open, trustable and reasonably efficient decision procedure for the logic of unquantified formulas over uninterpreted symbols, difference logic over integer and real numbers, full linear arithmetic on reals, and the combination thereof. This corresponds to the logics identified as QF_IDL, QF_RDL, QF_LRA, QF_UF, QF_UFLRA and QF_UFIDL in the SMT-LIB benchmarks [34,4]. Internally, the solver is organized to be easily extended by plugging new decision procedures. Algorithm 2 is the kernel of the solver; it uses the MiniSAT solver [16] to produce models of the Boolean abstraction of the input formula. veriT is open-source and distributed under the BSD licence at http://www.verit-solver.org.

Although not (yet) as fast as the solvers performing best in the SMT competition [4], veriT has a decent efficiency. The tool does not yet implement theory propagation [29], a technique that is known to greatly improve the efficiency of SMT solvers. It participated in the annual SMT competition in 2009.[5]

In addition to giving a yes or no answer, the veriT solver produces proofs. Proof production has two goals. First, this feature increases the confidence in the tool, the proofs being checked by an independent module inside veriT. Second, skeptical proof assistants can use such traces to reconstruct the proofs of formulas discharged by veriT (see [18]). Combining decision procedure by (model-)equality exchange is particularly suitable for generating proofs. Indeed, as showed in the two illustrating examples in Sections 6 and 7, introducing model equalities does not involve another reasoning process, but literals that are not assigned by the SAT-solver may appear in conflict clauses. The proof remains essentially the resolution proof from the SAT solver.

[5] The results are available on the web at http://www.smtcomp.org/.

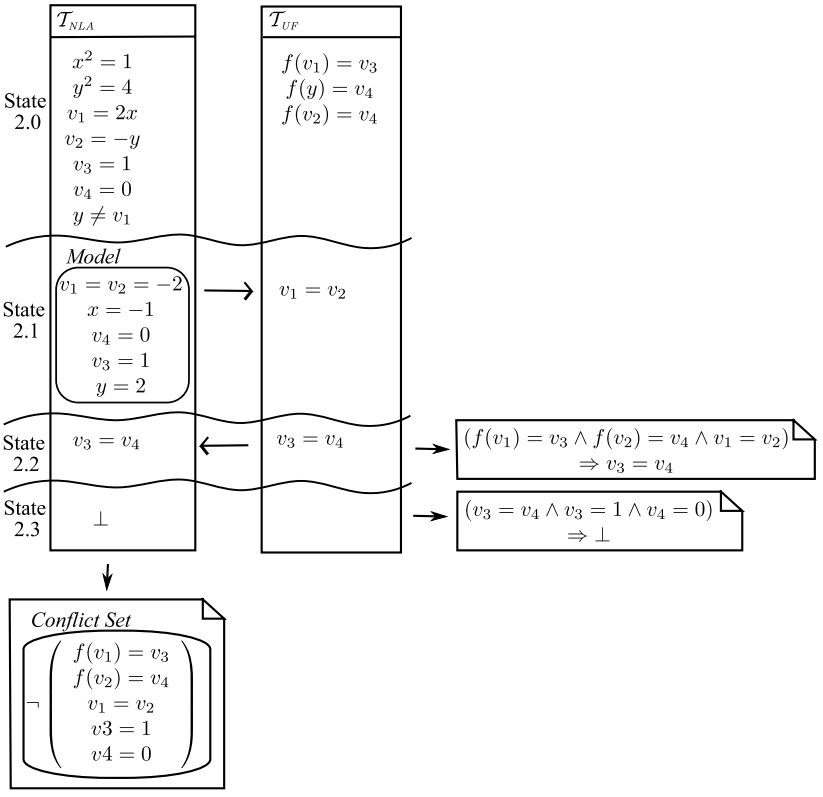

Fig. 4. Example UF and NLA: trying the second assignment.

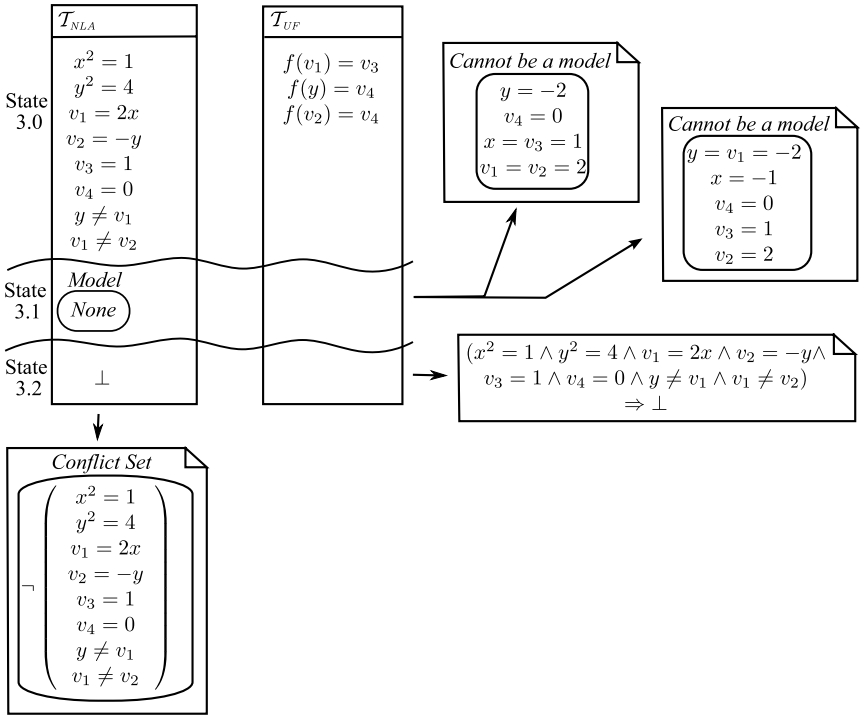

Fig. 5. Example UF and NLA: Trying the third and last assignment.

9. Conclusion

In this paper, we presented an approach to combine decision procedures, based on the exchange of equalities only. The originality is that the combination maintains the completeness property of the classic approach if the decision procedures have the capability to generate model equalities instead of disjunctions of equalities. In practice, this new requirement is often much simpler to implement than the original one. We also proposed a simple abstract framework to reason about SMT-solvers and applied it to show the completeness of our approach based on the generation and propagation of model equalities. This approach inherits model-based guessing from [13], and the interaction of decision procedures through the SAT-solver from [5]. It differs from [13] in the fact that model-based guessing is now integrated in a classical Nelson–Oppen equality exchange, seeing it just as a new way to exchange equalities. The SAT-solver realizes the case-splitting, and this guarantees completeness if every decision procedure in the combination is either convex and produces all deducible equalities or is able to produce model-equalities.

We aim to provide an implementation that is easily integrated in other deduction tools [18] such as Isabelle [30], Coq [21], HOL [19] or HOL-light [20]. For this, it is necessary to provide detailed proofs. The present approach is particularly suitable, since it makes a clear distinction between the propositional reasoning and the theory reasoning. Propositional reasoning steps as well as equality propagating steps consist in propositional resolution steps. The reasoning steps for equality deduction and conflicts are specific to theories.

Since the bottleneck of the cooperation between proof assistants and SMT-solvers is not the efficiency of the SMT-solvers, our focus has not been much directed towards the efficiency of our implementation. In particular, we do not implement theory propagation (see for instance [29]). However, we observed that for the specific QF_UFIDL benchmarks reported to work particularly well for [13], veriT works as fast as in Z3 [14]. In general, the performances of our tool are decent compared to the other state-of-the-art tools, as attested by the results of the SMT-COMP'2009.

Current and future works include applying this technique to other decidable fragments (for instance full linear arithmetic on integer and real numbers). Also, our implementation includes a full-featured first-order theorem prover that handles user theories. We will then investigate the benefits of our framework in presence of such user defined theories.

Acknowledgements

We are grateful to the anonymous reviewers for their remarks.

References

[1] M. Abadi, L. Lamport, The existence of refinement mappings. Technical Report 29, 1988, DEC/SRC.
[2] J.-R. Abrial, The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.
[3] M. Baaz, U. Egly, A. Leitsch, Normal form transformations, in: J.A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning. Vol. I, Elsevier Science B.V., 2001, pp. 273–333 (Ch. 5).
[4] C. Barrett, M. Deters, A. Oliveras, A. Stump, Design and results of the 3rd annual satisfiability modulo theories competition (SMT-COMP 2007), International Journal on Artificial Intelligence Tools 17 (4) (2008) 569–606.
[5] C. Barrett, R. Nieuwenhuis, A. Oliveras, C. Tinelli, Splitting on demand in SAT modulo theories, in: M. Hermann, A. Voronkov (Eds.), Proc. 13th Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR), in: Lecture Notes in Computer Science, Vol. 4246, Springer, 2006, pp. 512–526.
[6] C. Barrett, R. Sebastiani, S.A. Seshia, C. Tinelli, Satisfiability modulo theories, in: A. Biere, M.J.H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, in: Frontiers in Artificial Intelligence and Applications, Vol. 185, IOS Press, 2009, pp. 825–885 (Ch. 26).
[7] C.W. Barrett, D.L. Dill, A. Stump, A generalization of Shostak's method for combining decision procedures, in: A. Armando (Ed.), Frontiers of Combining Systems, FroCoS, in: Lecture Notes in Computer Science, Vol. 2309, Springer, 2002, pp. 132–146.
[8] D. Barsotti, L.P. Nieto, A. Tiu, Verification of clock synchronization algorithms: experiments on a combination of deductive tools, Formal Aspects of Computing 19 (3) (2007) 321–341.
[9] T. Bouton, D.C.B. de Oliveira, D. Déharbe, P. Fontaine, veriT: an open, trustable and efficient SMT-solver, in: R. Schmidt (Ed.), Proc. Conference on Automated Deduction (CADE), in: Lecture Notes in Computer Science, Vol. 5663, Springer, Montreal, Canada, 2009, pp. 151–156.
[10] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, Delayed theory combination vs. Nelson–Oppen for satisfiability modulo theories: a comparative analysis, in: M. Hermann, A. Voronkov (Eds.), Proc. 13th Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR), in: Lecture Notes in Computer Science, Vol. 4246, Springer, 2006, pp. 527–541.
[11] M. Cornélio, A. Cavalcanti, A. Sampaio, Refactoring towards a layered architecture, in: A. Moura (Ed.), Proc. Brazilian Symposium on Formal Methods, SBMF 2004, 2004, pp. 199–216.
[12] M. Davis, G. Logemann, D. Loveland, A machine program for theorem proving, Communications of the ACM 5 (7) (1962) 394–397.
[13] L. de Moura, N. Bjørner, Model-based theory combination, Electronic Notes in Theoretical Computer Science 198 (2) (2008) 37–49.
[14] L. de Moura, N. Bjørner, Z3: An efficient SMT solver, in: C. Ramakrishnan, J. Rehof (Eds.), Proc. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), in: Lecture Notes in Computer Science, Vol. 4963, Springer, 2008, pp. 337–340.
[15] D.C.B. de Oliveira, D. Déharbe, P. Fontaine, Combining decision procedures by (model-)equality propagation, in: Proceedings of the Eleventh Brazilian Symposium on Formal Methods, SBMF 2008, Salvador, Brazil, 26–29 August 2008, Electronic Notes in Theoretical Computer Science 240 (2009) 113–128.
[16] N. Eén, N. Sörensson, An extensible SAT-solver, in: E. Giunchiglia, A. Tacchella (Eds.), Theory and Applications of Satisfiability Testing, in: Lecture Notes in Computer Science, Vol. 2919, Springer, 2003, pp. 333–336. 6th International Conference, SAT 2003.
[17] P. Fontaine, E.P. Gribomont, Combining non-stably infinite, non-first order theories, in: W. Ahrendt, P. Baumgartner, H. de Nivelle, S. Ranise, C. Tinelli (Eds.), Selected Papers from the Workshops on Disproving and the Second International Workshop on Pragmatics of Decision Procedures (PDPAR 2004), in: Electronic Notes in Theoretical Computer Science, Vol. 125, 2005, pp. 37–51.
[18] P. Fontaine, J.-Y. Marion, S. Merz, L.P. Nieto, A. Tiu, Expressiveness + automation + soundness: towards combining SMT solvers and interactive proof assistants, in: H. Hermanns, J. Palsberg (Eds.), Tools and Algorithms for Construction and Analysis of Systems (TACAS), in: Lecture Notes in Computer Science, Vol. 3920, Springer, 2006, pp. 167–181.
[19] M.J.C. Gordon, T.F. Melham (Eds.), Introduction to HOL: A Theorem Proving Environment for Higher Order Logic, Cambridge University Press, 1993.

[20] J. Harrison, HOL light: a tutorial introduction, in: M. Srivas, A. Camilleri (Eds.), Proc. First Int'l Conf. on Formal Methods in Computer-Aided Design (FMCAD'96), in: Lecture Notes in Computer Science, Vol. 1166, Springer, 1996, pp. 265–269.
[21] G. Huet, G. Kahn, C. Paulin-Mohring, The Coq Proof Assistant — a tutorial, 2004, Version 8.0.
[22] S.K. Lahiri, M. Musuvathi, An efficient decision procedure for UTVPI constraints, in: B. Gramlich (Ed.), Proc. Int'l Workshop Frontiers of Combining Systems, FroCoS 2005, in: Lecture Notes in Computer Science, Vol. 3717, Springer, 2005, pp. 168–183.
[23] S.K. Lahiri, M. Musuvathi, An efficient Nelson–Oppen decision procedure for difference constraints over rationals, Electronic Notes in Theoretical Computer Science 144 (2) (2006) 27–41.
[24] K.R.M. Leino, Object invariants in specification and verification, in: A.M. Moreira, L. Ribeiro (Eds.), Proc. Brazilian Symposium on Formal Methods, SBMF 2006, 2006, pp. 3–4.
[25] C. Morgan, Programming from Specifications, Prentice Hall International, 1994.
[26] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: engineering an efficient SAT solver, in: Proceedings of the 38th Design Automation Conference, DAC'01, 2001, pp. 530–535.
[27] G. Nelson, D. Oppen, Simplification by cooperating decision procedures, ACM Transactions on Programming Languages and Systems 1 (2) (1979) 245–257.
[28] G. Nelson, D. Oppen, Fast decision procedures based on congruence closure, Journal of the ACM 27 (2) (1980) 356–364.
[29] R. Nieuwenhuis, A. Oliveras, DPLL(T) with exhaustive theory propagation and its application to difference logic, in: K. Etessami, S. Rajamani (Eds.), Computer Aided Verification (CAV), in: Lecture Notes in Computer Science, Vol. 3576, Springer, 2005, pp. 321–334.
[30] T. Nipkow, L. Paulson, M. Wenzel, Isabelle/HOL, in: A Proof Assistant for Higher-Order Logic, in: Lecture Notes in Computer Science, Vol. 2283, Springer, 2002.
[31] A. Nonnengart, C. Weidenbach, Computing small clause normal forms, in: J.A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning. Vol. I, Elsevier Science B.V., 2001, pp. 335–367 (Ch. 6).
[32] D.C. Oppen, Complexity, convexity and combinations of theories, Theoretical Computer Science 12 (3) (1980) 291–302.
[33] V. Pratt, Two easy theories whose combination is hard, 1977. http://citeseer.ist.psu.edu/pratt77two.html.
[34] S. Ranise, C. Tinelli, The SMT-LIB standard: Version 1.2, 2006.
[35] R.E. Shostak, Deciding combinations of theories, Journal of the ACM 31 (1) (1984) 1–12.
[36] C. Tinelli, M.T. Harandi, A new correctness proof of the Nelson–Oppen combination procedure, in: F. Baader, K.U. Schulz (Eds.), Proc. Frontiers of Combining Systems, FroCoS. Applied Logic, Kluwer Academic Publishers, 1996, pp. 103–120.
[37] C. Tinelli, C.G. Zarba, Combining nonstably infinite theories, Journal of Automated Reasoning 34 (3) (2005) 209–238.
[38] L. Zhang, C. Madigan, M. Moskewicz, S. Malik, Efficient conflict driven learning in boolean satisfiability solver, in: Proc. Int'l Conf. on Computer Aided Design, ICCAD, 2001, pp. 279–285.