Combining decision procedures by (model-)equality propagation[*]

Diego Caminha B. de Oliveira[2], David Déharbe[1] and Pascal Fontaine[2]

[1] UFRN - CCET - DIMAp
Natal – RN – Brazil

[2]Université de Nancy – INRIA – LORIA
Nancy – France

{Diego.Caminha,Pascal.Fontaine}@loria.fr, david@dimap.ufrn.br

Abstract. *SMT (Satisfiability Modulo Theories) solvers are automatic verification engines suitable to discharge important classes of proof obligations generated in applying formal construction of software and hardware designs. In this paper, we present a new approach to combine decision procedures and propositional solvers into an SMT-solver. This approach is based on the generation of model equalities by decision procedures. We show the soundness and completeness of the proposed approach using an original abstract framework to represent and reason about SMT-solvers. We then present an algorithmic version of the new SMT-solving approach and discuss practical aspects of our implementation.*

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 high order logic, but also process algebra and temporal logic. 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 arithmetics, set and array theories, etc. This applies to a number of verification applications, e.g. the application of formal program transformations such as refinement [Morgan 1994] or refactoring laws [Cornélio et al. 2004], verification of refinement properties in *posit-and-prove* software engineering efforts [Abadi and Lamport 1988, Abrial 1996], or static analysis of annotations in design-by-contract languages [Leino 2006]. Even verification efforts in more expressive logics often require proving lemmas that may be tackled by SMT-solvers (see for instance [Barsotti et al. 2007]).

SMT-solvers can for example handle a formula like

$$x \leq y \,\wedge\, y \leq x + f(x) \,\wedge\, P(h(x) - h(y)) \,\wedge\, \neg P(0) \,\wedge\, f(x) = 0 \qquad (1)$$

[*]The research presented in this paper has been partially financed by CNPq/INRIA project Da Capo and CNPq project N° 307597/2006-7.

which contains linear arithmetics on reals (0, $+$, $-$, $\leq$), and uninterpreted symbols (P, h, f). SMT-solvers use decision procedures for the disjoint languages (for instance, congruence closure for uninterpreted symbols [Nelson and Oppen 1980], and simplex for linear arithmetics) 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. In the general case, the information exchanged between the decision procedures is a set of disjunctions of equalities, and handling them requires often complex and costly case splitting. In the special case of convex theories, exchanging only equalities (and not disjunctions) is enough to ensure the completeness of the combination. We here show that even in the general case (i.e. with non-convex theories) exchanging equalities is also sufficient for completeness thanks to the cooperation with the propositional reasoning engine of the SMT-solver.

The next section introduces notations and the basics of SMT-solvers. In Section 3 we present an abstract framework for describing SMT-solvers. It only serves to discuss the soundness and completeness of the combination framework we describe in this paper. It is not as detailed as the DPLL($\mathcal{T}$) framework [Nieuwenhuis and Oliveras 2005] since it is not meant to be a precise description of solvers. By contrast to the DPLL($\mathcal{T}$) framework and for simplicity, our schema highlights the distinction between the Boolean reasoning and the theory reasoning. It is not difficult to understand DPLL($\mathcal{T}$) as being a refinement of our schema.

Section 4 uses the framework introduced in Section 3 to discuss the soundness and completeness of various approaches to SMT solving. In particular, we present a new approach that consists in only exchanging equalities: either those equalities are deduced by the decision procedures, or they are assumed by generalising models. The approach is suitable for any decision procedure capable of finding models; many decision procedures inherently have this capability. A concrete algorithm using this approach is presented in Section 5. This algorithm is a simplification of our implementation. A concrete example is discussed in Section 6.

2. Basics of SMT-solvers

2.1. Notations

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. Every function and predicate symbol is assigned an *arity*. 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 *theory* is a set of closed formulas. Two theories are *disjoint* if no predicate symbol in $\mathcal{P}$ or function symbol in $\mathcal{F}$ is interpreted 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.

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 always possible to transform a quantifier-free formula into equivalent or equisatisfiable conjunctive normal form. 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 $x_1 = y_1 \vee \dots x_n = y_n$ is a logical consequence of a set of literals Γ (i.e. $\Gamma \models_{\mathcal{T}} x_1 = y_1 \vee \dots x_n = y_n$), 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 model.

A *propositional abstraction* for a set of formulas is this set of formulas in which every atom has been replaced by a proposition, every occurrence 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 φ. For instance, consider φ is the following formula:

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

Each atom of φ is abstracted to a propositional variable p_i. In practice, propositional SAT-solvers represent formulas in conjunctive normal form; 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 unsatisfiable modulo theory. 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 these literals needs then to be verified. In general, the literals may contain symbols from several theories $\mathcal{T}_1, \mathcal{T}_2, \ldots \mathcal{T}_k$. 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 [Shostak 1984] or Nelson and Oppen's [Nelson and Oppen 1979].

In the Nelson and Oppen framework, the set of literals is used to produce an equi-satisfiable set $\Gamma' = \Gamma_1 \cup \Gamma_2 \cup \ldots \Gamma_k$ 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 set of literals for (1) and for the set of literals corresponding to the former entailing assignment $\{p_1, p_2, p_3, p_4, \neg p_5\}$ can be:

$$\begin{aligned} \Gamma_1 &= \{x \leq y,\, y \leq x + \mathbf{v_1},\, \mathbf{v_1} = 0,\, \mathbf{v_2} = \mathbf{v_3} - \mathbf{v_4},\, \mathbf{v_5} = 0\} \\ \Gamma_2 &= \{P(\mathbf{v_2}),\, \neg P(\mathbf{v_5}),\, \mathbf{v_1} = f(x),\, \mathbf{v_3} = h(x),\, \mathbf{v_4} = h(y)\} \end{aligned}$$

where v_1 to v_5 are new variables, $\mathcal{T}_1$ and $\mathcal{T}_2$ are respectively the theories for linear arithmetics on reals and for uninterpreted functions. One then identifies the set of shared variables $\mathcal{S} = \{v_1, v_2, v_3, v_4, v_5, x, y\}$. The decision procedures for each theory $\mathcal{T}_i$ receive the corresponding set of now pure literals Γ_i and should propagate to other decision procedures all the equalities between the shared variables that can be derived from Γ_i and from such equalities received from the other decision procedure. If the signatures are disjoint and the theories stably infinite, then the original set of literals is unsatisfiable if and only if unsatisfiability is derived by one of the decision procedures.

If the original set of literals Γ is satisfiable, then it is a model modulo theory of the original formula φ, which is itself satisfiable. Otherwise, the propositional solver needs to be updated to prevent it from generating such assignment again. From a logical viewpoint this corresponds to adding the negation of the assignment, or any unsatisfiable subset thereof. 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 such subset. 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.

In some cases, the decision procedures, or the combination framework, may be able to generate lemmas from φ that may be useful to guide 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 arithmetics property at the propositional level.

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. The SAT-solver maintains a propositional assignment Γ for this set of clauses. The pair S, Γ thus represents the current state of the solver. The set of rules given in

$$\textsc{Bool}: \quad \frac{S, \Gamma}{S, \Gamma'} \qquad \Gamma' \text{ is a propositional assignment of } S \tag{2}$$

$$\textsc{Unsat}: \quad \frac{S, \Gamma}{\text{UNSAT}} \qquad S \text{ is propositionally unsatisfiable} \tag{3}$$

$$\textsc{Learn}: \quad \frac{S, \Gamma}{S \cup C, \Gamma} \qquad S \models_{\mathcal{T}} C \tag{4}$$

$$\textsc{Sat}: \quad \frac{S, \Gamma}{\text{SAT}(\Gamma)} \qquad \Gamma \text{ is entailing and } \mathcal{T}\text{-satisfiable} \tag{5}$$

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

Figure 1 schematizes the possible steps taken by the solver and are described in details in the following. Observe that clauses may be added to the set 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 is also $\mathcal{T}$-satisfiable.

Rule BOOL (2) 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 (3).

The addition of new clauses is represented by rule LEARN (4). 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 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 equisatisfiable to the original set of clauses.

If the assignment produced by the SAT-solver is entailing and $\mathcal{T}$-satisfiable, then the theory solver may conclude that the formula is satisfiable. This is summarized in rule SAT (5).

In the present scheme, no assumption is made on the order of application of rules, on how the clause C is generated in LEARN (4) 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 Figure 1 is sound.*

Proof: Every clause added to the initial set of clauses using rule LEARN (4) is a logical consequence of the initial set of clauses. If the SAT-solver concludes to the propositional unsatisfiability of the set of clauses, the initial set of clauses is unsatisfiable. Also, if there is a propositional entailing assignment that is $\mathcal{T}$-satisfiable, the original set of clauses is satisfiable. □

Notice that the assumption in rule LEARN (4) is very permissive. It holds notably for propositional learning, where the new clause C is obtained by propositional resolution of clauses in S, guided by the FUIP computation [Zhang et al. 2001]. 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 Figure 1.

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

- *if the set of clauses remains unchanged*[1]*, the SAT-solver will eventually either*
  - *provide an entailing assignment*
  - *or conclude to the unsatisfiability of the set of clauses with rule* UNSAT *(3);*
- *the atoms of the clauses added in rule* LEARN *(4) belong to a finite set that is fixed* a priori *for the whole run of the SMT-solver;*
- *for any state S, Γ where Γ is entailing, either rule* SAT *(5) is applied or rule* LEARN *(4) 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 (5) or rule UNSAT (3). 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 S, Γ. Since the set of clauses S does not change, 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 (5)) or
- the rule LEARN (4) is applied and introduces a clause C that is not a propositional consequence of Γ.

The first option is not possible, since the ending state is S, Γ. The second option is also not possible, since this would change the set S, and contradict the fact that S, Γ is the ending state.

Assume now that the run is infinite. The set of atoms that are 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 (4), and the SAT-solver will eventually provide an entailing assignment Γ. The rule SAT (5) being an ending rule, the next rule will be rule LEARN (4), 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; if the set of clauses does not change, SAT-solvers will eventually decide that the set of clauses is unsatisfiable, or provide a total (and thus entailing) assignment. The two remaining requirements are related to the theory reasoner and are discussed in the next section.

4. Combination of theories and propositional reasoning

In this section, we instantiate the previous framework to common approaches of combinations of theories, and discuss those various approaches.

[1] By unchanged, we mean that no clause is added, or every added clause is already in the set of clauses known by the SAT-solver.

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). An arrangement $\mathcal{A}$ of a set of variables is a set that contains either $X = Y$ or $X \neq Y$ for every pair of variables X, Y in the set. For instance, an arrangement for the set of variables $\mathcal{S} = \{v_1, v_2, v_3, v_4, v_5, x, y\}$ from the example in Section 2 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 (dis)equalities have been ignored). The result behind the Nelson-Oppen combination scheme (see for instance [Nelson and Oppen 1979, Tinelli and Harandi 1996]) states that Γ (or equivalently $\Gamma_1 \cup \Gamma_2$) is $\mathcal{T}$-satisfiable if and only if there exists an arrangement $\mathcal{A}$ of the shared variables such that $\mathcal{A} \cup \Gamma_1$ is $\mathcal{T}_1$-satisfiable and $\mathcal{A} \cup \Gamma_2$ is $\mathcal{T}_2$-satisfiable. The $\mathcal{T}$-satisfiability problem is thus reduced to a set of $\mathcal{T}_1$-satisfiability and $\mathcal{T}_2$-satisfiability problems.

There are as many different arrangements for a set of variables as partitions of that set. The number of arrangements 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 [Bruttomesso et al. 2006]) 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 assignment is then given to every theory reasoner in the combination; each theory reasoner only picks among the assignment the literals that are relevant to the theory. This technique is implemented in several state of the art SMT-solvers. When given a total assignment Γ, one theory reasoner 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. In the other case, every theory reasoner concludes that its set is satisfiable, and Γ is also satisfiable in the combination of theories since every (entailing) total assignment contains an arrangement. 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 deduced disjunctions of equalities from the independent theories. Informally, the following theorem states that two disjoint theories can agree on the satisfiability of a set of literals if no disjunction of equality can be deduced by one or the other theory (and given as a new fact to the other theory). 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).

[2] Disjointness and stably infiniteness are sufficient conditions for the easy combination of two theories. Those conditions are not necessary, but for simplicity, we assume those conditions are met.

[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.

Theorem 3 *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 $\Delta = (x_1 = y_1 \vee \ldots x_n = y_n)$ (where $x_1, \ldots x_n$ and $y_1, \ldots y_n$ 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 bet the set of all equalities $x = y$ such that $\Gamma_1 \models_{\mathcal{T}_1} x = y$, and S_2 bet the set of all equalities $x = y$ such that $\Gamma_2 \models_{\mathcal{T}_2} x = y$. Assume $x_1 = y_1$ 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 the result behind the Nelson-Oppen combination scheme $\mathcal{A} \cup \Gamma_1$ 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 $\mathcal{A} \cup \Gamma_1$ 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 $\Gamma_1 \models_{\mathcal{T}_1} x_1 = y_1 \vee \ldots x_n = y_n$. 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 $x'_1 = y'_1 \vee \ldots x'_{n'} = y'_{n'}$ such that $\Gamma_2 \cup \{x_i = y_i\} \models x'_1 = y'_1 \vee \ldots x'_{n'} = y'_{n'}$. 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) are 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 realised 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. As a direct consequence, no splitting is required when combining convex theories only. Many useful theories are convex, and notably linear arithmetics on the reals, and the theory of uninterpreted functions. Among the non-convex theories one finds the theory of linear arithmetics 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* [Barrett et al. 2006]. In such a case, rather than splitting on a disjunction $x_1 = y_1 \vee \ldots x_n = y_n$ that is a $\mathcal{T}_i$-logical consequence of the set of literals Γ_i, a new clause $x_1 = y_1 \vee \ldots x_n = y_n \vee \bigvee_{\ell \in \Gamma_i} \neg \ell$ 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 added to 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 theory reasoners, handling case splitting through the SAT-solver may introduce new atoms, namely equalities between terms that are not in the original formula. However, since there are 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 3 such a clause exists whenever the assignment is $\mathcal{T}$-unsatisfiable; this clause is not a propositional consequence of the assignment. If the theory reasoners 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 arithmetics over the integer) it is not easy to be complete for the deduction of disjunction of equalities (see for instance [Lahiri and Musuvathi 2005]). Moreover some decision procedures for convex theories are not easily tweaked to produce equalities between variables. The next section presents another way to ensure completeness of SMT-solvers in those cases.

4.3. Introducing model equalities

The method we present here 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 reasoner for integer linear arithmetics 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. Assume also 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. Those 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 (5) can be applied and the third assumption of Theorem 2 is fulfilled. The approach is sound and complete.

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

Algorithm 1 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 (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 consecutive sets of literals produced by the propositional SAT-solver as they may update their state to reflect only the difference between these sets.

As described in Section 4.3, we also assume that the decision procedures for the theories $\mathcal{T}_i$ are able to generate the *model equalities* based on a model they keep based on the literals, equalities and inequalities they receive. The main difference with respect to a version without model equalities is located in the lines 17 to 21.

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 (3)). Otherwise, a propositional model is computed (line 2) (Rule BOOL (2)). 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 *newLiterals* will maintain 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 16 and 19). This set is repeatedly propagated to each decision procedure until one of them detects unsatisfiability (line 9) or no new equalities can be deduced (line 22). 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 (4). Otherwise, each decision procedure computes the set of variable equalities entailed by the current set of literals. These sets are stored for propagation at the next iteration (line 16). 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[4].

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 23–25). Again, this corresponds to an instance of Rule LEARN (4). Eventually, when no new information can be provided to the propositional satisfiability

[4] In this algorithm, we assume that none of the non-convex theories have capabilities of generating disjunctions of equalities, i.e., internal case split is not handled.

solver, and if the assignment is total, then the algorithm concludes that the original formula is $\mathcal{T}$-satisfiable and halts (line 26), which corresponds to Rule SAT (5).

```
 1 while $satSolver.status() = $ SAT do
 2     $assignment := satSolver.assignment()$;
 3     $status :=$ SAT;
 4     foreach t in theories do  $t.backJump(assignment)$;
 5     $newLiterals := assignment.getNewLiterals()$;
 6     repeat
 7         foreach t in theories do
 8             $status := t.propagateLiterals(newLiterals)$;
 9             if $status =$ UNSAT then
10                 $satSolver.add(t.conflictClause)$;
11                 $break$;
12             end
13         end
14         if status = UNSAT then break;
15         $newLiterals := \{\}$;
16         foreach t in theories do  $newLiterals$ += $t.getNewEqualities()$;
17         if $newLiterals = \{\}$ then
18             foreach t in theories $\wedge$ t.deduction $\neq$ COMPLETE do
19                 $newLiterals$ += $t.getNewModelEqualities()$;
20             end
21         end
22     until newLiterals = {} ;
23     $lemmas := \{\}$;
24     foreach t in theories do  $lemmas$ += $t.getNewLemmas()$;
25     $satSolver.add(lemmas)$;
26     if lemmas = {} $\wedge$ status = SAT $\wedge$ assignment = TOTAL then  return SAT;
27 end
28 return UNSAT;
```

Algorithm 1: Satisfiability Check

6. Combining Uninterpreted Functions with Integer Difference Logic: An Example

In this section we present an example of how to handle the combined theory of uninterpreted functions (UF) and integer difference logic (IDL) using *model equalities*, i.e., not having to generate disjunction of equalities necessary for completeness in a Nelson and Oppen (NO) 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 $\bowtie \in \{\leq, \geq, =, <, >\}$. We want to prove that the formula φ is unsatisfiable.

$$\varphi : x \leq y + 1 \wedge y \leq x \wedge x \neq y \wedge f(x) \neq f(y+1) \tag{6}$$

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. The obtained formula is φ', and each different atom is attributed to a propositional variable p_i.

$$\varphi' : \overbrace{v_1 = y + 1}^{p_1} \wedge \overbrace{x \leq v_1}^{p_2} \wedge \overbrace{y \leq x}^{p_3} \wedge \overbrace{x \neq y}^{\neg p_4} \wedge \overbrace{f(x) \neq f(v_1)}^{\neg p_5} \tag{7}$$

Figure 2 can be used to trace the status of the algorithm during its application to this problem. We use the symbols p_i to represent the constraints and compact the figure. We also consider that $\neg(a = b)$ is $a \neq b$, $\neg(a \leq b)$ is $a > b$, and $\neg(a \geq b)$ is $a < b$.

	Formula	Assignment	Decision procedures
1	$p_1 \wedge p_2 \wedge p_3 \wedge \neg p_4 \wedge \neg p_5$	$\{ p_1, p_2, p_3,$ $\neg p_4, \neg p_5 \}$	$\mathcal{T}_1 : \{\neg p_4, \neg p_5\}$ $\mathcal{T}_2 : \{p_1, p_2, p_3, \neg p_4\}$
1.1			$\mathcal{T}_1 : \{\neg p_4, \neg p_5, p_6\}$ $\mathcal{T}_2 : \{p_1, p_2, p_3, \neg p_4, p_6\}$
2	$p_1 \wedge p_2 \wedge p_3 \wedge \neg p_4 \wedge \neg p_5$ $\wedge(p_5 \vee \neg p_6)$	$\{ p_1, p_2, p_3,$ $\neg p_4, \neg p_5, \neg p_6 \}$	$\mathcal{T}_1 : \{\neg p_4, \neg p_5, \neg p_6\}$ $\mathcal{T}_2 : \{p_1, p_2, p_3, \neg p_4, \neg p_6\}$
3	$p_1 \wedge p_2 \wedge p_3 \wedge \neg p_4 \wedge \neg p_5$ $\wedge(p_5 \vee \neg p_6)$ $\wedge(p_6 \vee \neg p_2 \vee \neg p_7)$	$\{ p_1, p_2, p_3, \neg p_4,$ $\neg p_5, \neg p_6, \neg p_7 \}$	$\mathcal{T}_1 : \{\neg p_4, \neg p_5, \neg p_6\}$ $\mathcal{T}_2 : \{p_1, p_2, p_3, \neg p_4, \neg p_6, \neg p_7\}$
3.1			$\mathcal{T}_1 : \{\neg p_4, \neg p_5, \neg p_6, p_4\}$ $\mathcal{T}_2 : \{p_1, p_2, p_3, \neg p_4, \neg p_6, \neg p_7, p_4\}$
4	$p_1 \wedge p_2 \wedge p_3 \wedge \neg p_4 \wedge \neg p_5$ $\wedge(p_5 \vee \neg p_6)$ $\wedge(p_6 \vee \neg p_2 \vee \neg p_7)$ $\wedge(p_4 \vee p_7 \vee \neg p_1 \vee \neg p_3)$ $\Rightarrow$ UNSAT		

Figure 2. An example combining UF and IDL.

State 1 corresponds to the computation by the SAT-solver of a propositional assignment Γ of the formula. The constraints from Γ are propagated to the decision procedures to check for theory consistency. $\mathcal{T}_1$ and $\mathcal{T}_2$ are the decision procedures for UF and IDL, respectively. At this point, no equality can be generated and the process would end if there were only convex theories involved.

However, IDL is non-convex. Two approaches are possible here: generate a disjunction of equalities, as proposed in Section 4.2 or a model equality, a new approach proposed in Section 4.3 and adopted in Algorithm 1. A difference logic solver based on graph algorithms (see e.g. [Lahiri and Musuvathi 2006]) can generate one model easily. In a consistent model we would have values such as $x = 0$, $y = -1$ and $v_1 = 0$, so the *model equality* $x = v_1$ (abstracted to proposition p_6) is generated and propagated to all decision procedures. State 1.1 is then reached, and a contradiction is found in the UF engine, as 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 and a second iteration of the main loop is necessary. The algorithm reaches State 2, where a new assignment is generated. The decision procedures backtrack to the previous satisfiable state, i.e., just before propagating the model equality $x = v_1$ (i.e. p_6). The set of literals is thus incremented with literal $\neg p_6$ (i.e. $x \neq v_1$) which is dispatched to both decision procedures. No contradiction or equality is generated, but IDL finds that the current assignment conflicts with the previously generated model equality. To remedy this situation, the IDL decision procedure generates a *lemma*: $(x = v_1) \vee (x > v_1) \vee (x < v_1)$. The lemma is incorporated to the SAT-solver. This results in the creation of a new propositional variable p_7 corresponding to the atom $v_1 \leq x$.

The lemma is incorporated by the SAT-solver and the main loop is thus iterated a third time, the algorithm reaching State 3. A new assignment is generated, resulting in the addition of a new literal $\neg p_7$ (i.e. $\neg v_1 \leq x$, or $x < v_1$), which is propagated to the decision procedure for IDL. This decision procedure is then in condition to deduce the *equality between shared variables* $x = y$ from $x < v_1$, $v_1 = y + 1$ and $y \leq x$. The algorithm reaches then State 3.1, where this equality has been 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.

At that point, State 4, the SAT-solver concludes there is no more assignment that makes the current formula propositionally true. Therefore, the problem is unsatisfiable.

7. Conclusion

In this paper, we presented a new approach to combine decision procedures, based on the generation of model equalities. 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 disjunction 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 [de Moura and Bjørner 2008a], and the interaction of decision procedures through the SAT-solver from [Barrett et al. 2006].

We aim to provide an implementation that is easily integrated in other deduction tools such as Isabelle [Nipkow et al. 2002], Coq [Huet et al. 2004], HOL [Gordon and Melham 1993] or HOL-light [Harrison 1996]. 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 [Nieuwenhuis and Oliveras 2005]). However, we observed that for the specific QF_UFIDL benchmarks reported to work particularly well for [de Moura and Bjørner 2008a], our implementation works as fast as in Z3 [de Moura and Bjørner 2008b].

Future works include applying this technique to other decidable fragments (for instance full linear arithmetic on integer and reals). 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.

References

Abadi, M. and Lamport, L. (1988). The existence of refinement mappings. Technical Report 29, DEC/SRC.

Abrial, J.-R. (1996). *The B-Book: Assigning Programs to Meanings*. Cambridge University Press.

Barrett, C., Nieuwenhuis, R., Oliveras, A., and Tinelli, C. (2006). Splitting on demand in SAT modulo theories. In Hermann, M. and Voronkov, A., editors, *Proc. 13th Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 4246 of *Lecture Notes in Computer Science*, pages 512–526. Springer.

Barsotti, D., Nieto, L. P., and Tiu, A. (2007). Verification of clock synchronization algorithms: experiments on a combination of deductive tools. *Form. Asp. Comput.*, 19(3):321–341.

Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., and Sebastiani, R. (2006). Delayed theory combination vs. nelson-oppen for satisfiability modulo theories: A comparative analysis. In Hermann, M. and Voronkov, A., editors, *Proc. 13th Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 4246 of *Lecture Notes in Computer Science*, pages 527–541. Springer.

Cornélio, M., Cavalcanti, A., and Sampaio, A. (2004). Refactoring towards a layered architecture. In *Proc. Brazilian Symposium on Formal Methods (SBMF 2004)*, pages 199–216.

de Moura, L. and Bjørner, N. (2008a). Model-based theory combination. *Electr. Notes Theor. Comput. Sci*, 198(2):37–49.

de Moura, L. and Bjørner, N. (2008b). Z3: An efficient SMT solver. In *Proc. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer.

Gordon, M. J. C. and Melham, T. F., editors (1993). *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press.

Harrison, J. (1996). HOL light: A tutorial introduction. In Srivas, M. and Camilleri, A., editors, *Proc. First International Conference on Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer-Verlag.

Huet, G., Kahn, G., and Paulin-Mohring, C. (2004). *The Coq Proof Assistant - A tutorial, Version 8.0.*

Lahiri, S. K. and Musuvathi, M. (2005). An efficient decision procedure for UTVPI constraints. In Gramlich, B., editor, *Proc. Int'l Workshop Frontiers of Combining Systems (FroCoS) 2005*, volume 3717 of *Lecture Notes in Computer Science*, pages 168–183. Springer.

Lahiri, S. K. and Musuvathi, M. (2006). An efficient Nelson-Oppen decision procedure for difference constraints over rationals. *Electr. Notes Theor. Comput. Sci*, 144(2):27–41.

Leino, K. R. M. (2006). Object invariants in specification and verification. In *Proc. Brazilian Symposium on Formal Methods (SBMF 2006)*, pages 3–4.

Morgan, C. (1994). *Programming from Specifications*. Prentice Hall International.

Nelson, G. and Oppen, D. (1979). Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257.

Nelson, G. and Oppen, D. (1980). Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364.

Nieuwenhuis, R. and Oliveras, A. (2005). DPLL(T) with exhaustive theory propagation and its application to difference logic. In Etessami, K. and Rajamani, S., editors, *Proc. 17th Int'l Conf. on Computer Aided Verification (CAV)*, volume 3576 of *Lecture Notes in Computer Science*, pages 321–334. Springer.

Nipkow, T., Paulson, L., and Wenzel, M. (2002). *Isabelle/HOL. A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag.

Shostak, R. E. (1984). Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12.

Tinelli, C. and Harandi, M. T. (1996). A new correctness proof of the Nelson–Oppen combination procedure. In Baader, F. and Schulz, K. U., editors, *Proc. Frontiers of Combining Systems (FroCoS)*, Applied Logic, pages 103–120. Kluwer Academic Publishers.

Zhang, L., Madigan, C., Moskewicz, M., and Malik, S. (2001). Efficient conflict driven learning in boolean satisfiability solver. In *Proc. Int'l Conf. on Computer Aided Design (ICCAD)*, pages 279–285.