On deciding satisfiability by DPLL($\Gamma + \mathcal{T}$) and unsound theorem proving

Maria Paola Bonacina[1]*, Christopher Lynch[2], and Leonardo de Moura[3]

[1] Dipartimento di Informatica, Università degli Studi di Verona
Strada Le Grazie 15, I-37134 Verona, Italy
mariapaola.bonacina@univr.it

[2] Department of Mathematics and Computer Science
Clarkson University, Potsdam, NY 13699-5815, U.S.A.
clynch@clarkson.edu

[3] Microsoft Research, One Microsoft Way, Redmond, WA 98052, U.S.A.
leonardo@microsoft.com

Abstract. Applications in software verification often require determining the satisfiability of first-order formulæ with respect to some background theories. During development, conjectures are usually false. Therefore, it is desirable to have a theorem prover that terminates on satisfiable instances. Satisfiability Modulo Theories (SMT) solvers have proven highly scalable, efficient and suitable for integrated theory reasoning. Superposition-based inference systems are strong at reasoning with equalities, universally quantified variables, and Horn clauses. We describe a calculus that tightly integrates Superposition and SMT solvers. The combination is refutationally complete if background theory symbols only occur in ground formulæ, and non-ground clauses are variable inactive. Termination is enforced by introducing additional axioms as hypotheses. The calculus detects any unsoundness introduced by these axioms and recovers from it.

1 Introduction

Applications in software verification have benefited greatly from recent advances in automated reasoning. Applications in this field often require determining the satisfiability of first-order formulæ with respect to some background theories. In numerous contexts in software verification, quantifiers are needed. For example, they are used for capturing frame conditions over loops, axiomatizing type systems, summarizing auxiliary invariants over heaps, and for supplying axioms of theories that are not already equipped with decision procedures for ground formulæ. Thus, many verification problems consist in determining the satisfiability of a set of formulæ $\mathcal{R} \uplus P$ modulo a background theory $\mathcal{T}$, where $\mathcal{R}$ is a set of non-ground clauses without occurrences of $\mathcal{T}$-symbols, and P is a large ground formula (or set of ground clauses) that may contain $\mathcal{T}$-symbols. The set

* Research supported in part by MIUR grant no. 2007-9E5KM8.

of formulæ $\mathcal{R}$ can be viewed as the axiomatization of an application specific theory. The background theory $\mathcal{T}$ is a combination of general-purpose theories commonly used in hardware and software verification, such as linear arithmetic and bit-vectors.

Satisfiability Modulo Theories (SMT) solvers have proven highly scalable, efficient and suitable for integrated theory reasoning. Most SMT solvers are restricted to ground formulæ, and integrate the Davis-Putnam-Logemann-Loveland procedure (DPLL) with satellite $\mathcal{T}_i$-solvers for ground satisfiability problems in special theories $\mathcal{T}_i$, $1 \leq i \leq n$, so that $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$. In comparison, superposition-based inference systems (SP) are strong at reasoning with equalities, universally quantified variables, and Horn clauses. Moreover, SP was proved to terminate and hence to be a satisfiability procedure for several theories of data structures [1, 2, 5].

The DPLL($\Gamma + \mathcal{T}$) calculus [11] integrates an SMT solver with an inference system Γ that is sound and refutationally complete for first-order logic with equality. The key to the integration is that the literals in the candidate model built by the DPLL engine can occur as premises of Γ-inferences. In general, the DPLL($\Gamma + \mathcal{T}$) calculus is not refutationally complete when $\mathcal{T}$ is not empty, even when $\mathcal{T}$-symbols do not occur in $\mathcal{R}$. For example, assume $\mathcal{R} = \{x = a \vee x = b\}$ and $P = \emptyset$, and the background theory $\mathcal{T}$ is arithmetic. The clause $\{x = a \vee x = b\}$ implies that any model has at most two elements, which is clearly incompatible with any model for arithmetic. A first contribution of this paper are the conditions under which DPLL($\Gamma + \mathcal{T}$) is refutationally complete when $\mathcal{T}$ is not empty.

DPLL($\Gamma + \mathcal{T}$) has to combine all the theories in $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$ and $\mathcal{R}$. Combination of theories in SMT solvers is usually done by the Nelson-Oppen scheme [20], which requires that each $\mathcal{T}_i$ be *stably infinite*[4] and its solvers capable of generating all entailed disjunctions of equalities between constants. The second requirement could be relaxed as in [12], if each $\mathcal{T}_i$-solver were able to generate a candidate model, which may not be the case in general for all $\mathcal{T}_i$-solvers and for Γ acting as $\mathcal{R}$-solver. A second contribution of this work is to explain how to apply known results on *variable inactivity* [1, 8] to combine the built-in theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$ and the axiomatized theory $\mathcal{R}$ in DPLL($\Gamma + \mathcal{T}$).

In software verification, during development time, several conjectures are false because of mistakes in the implementation or specification. Therefore, it is desirable to have a theorem prover that terminates on satisfiable instances. In general, this is not a realistic goal since pure first-order logic is not decidable, and, even worse, there is no sound and complete procedure for first-order logic formulæ of linear arithmetic with uninterpreted functions [15]. Axioms such as *transitivity* $(\neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq z) \vee x \sqsubseteq z)$ and *monotonicity* $(\neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y))$ are problematic for any resolution-based Γ, since they tend to generate an unbounded number of clauses, even with a selection function that selects negative literals to prevent self-resolutions. Such axioms may arise in formalizations of type systems for programming languages. The signature features a predicate $\sqsubseteq$

[4] Every $\mathcal{T}_i$-satisfiable ground formula has a model with domain of infinite cardinality.

that represents a subtype relationship, and a monadic function f that represents a type constructor, such as `Array-of`. As an example, assume that the axiomatization contains a monotonicity axiom $\neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y)$. Resolution with negative selection would generate an infinite sequence $\{f^i(a) \sqsubseteq f^i(b)\}_{i\geq 0}$ for each literal $a \sqsubseteq b$ in its input. In practice, it is seldom the case that we need to go beyond $f(a) \sqsubseteq f(b)$ or $f^2(a) \sqsubseteq f^2(b)$ to show satisfiability. A third and main contribution of this paper is a new calculus that combines DPLL($\mathit{\Gamma} + \mathcal{T}$) with *unsound theorem proving* [17] to avoid such infinitary behaviors and obtain decision procedures for axiomatizations relevant to software verification. The idea is to control the infinitary behavior by using additional hypotheses/axioms, detect any unsoundness they may introduce and recover from it.

2 Background

We employ basic notions from logic usually assumed in theorem proving. Let $\mathit{\Sigma}$ be a *signature* consisting of a set of *function* and *predicate* symbols, each with its *arity*, denoted by $\mathit{arity}(f)$, for symbol f. We call 0-arity function symbols *constant* symbols, and use a, b, c and d for constants, f, g, h for non-constant function symbols, and x, y, z for variables. We use $\simeq$ to denote the interpreted predicate symbol for equality and $\mathit{Var}(l)$ to denote the set of variables occurring in a term or literal l. A first order $\mathit{\Sigma}$-theory is presented, or axiomatized, by a set of $\mathit{\Sigma}$-sentences. We use the symbols $\mathcal{T}$ and $\mathcal{R}$ to denote such presentations. Interpreted symbols are those symbols whose interpretation is restricted to the models of a certain theory, whereas free or uninterpreted symbols are those symbols whose interpretation is unrestricted.

A $\mathit{\Sigma}$-structure $\mathit{\Phi}$ consists of a non-empty universe $|\mathit{\Phi}|$ and an interpretation for variables and symbols in $\mathit{\Sigma}$. For each symbol f in $\mathit{\Sigma}$, the interpretation of f is denoted by $\mathit{\Phi}(f)$. For a function symbol f with $\mathit{arity}(f) = n$, the interpretation $\mathit{\Phi}(f)$ is an n-ary function on $|\mathit{\Phi}|$ with $\mathsf{range}(\mathit{\Phi}(f)) = \{u \mid \exists v \in |\mathit{\Phi}|,\ \mathit{\Phi}(f)(v) = u\}$. For a predicate symbol p with $\mathit{arity}(p) = n$, $\mathit{\Phi}(p)$ is a subset of $|\mathit{\Phi}|^n$. The interpretation of a term t is denoted by $\mathit{\Phi}(t)$. If t is a variable or constant, $\mathit{\Phi}(t)$ is an element in $|\mathit{\Phi}|$. Otherwise, $\mathit{\Phi}(f(t_1, \ldots, t_n)) = \mathit{\Phi}(f)(\mathit{\Phi}(t_1), \ldots, \mathit{\Phi}(t_n))$. If S is a set of terms, $\mathit{\Phi}(S)$ means the set $\{\mathit{\Phi}(t) \mid t \in S\}$. Satisfaction $\mathit{\Phi} \models C$ is defined as usual, and if $\mathit{\Phi} \models C$, the structure $\mathit{\Phi}$ is said to be a model of C.

An *inference system* $\mathit{\Gamma}$ is a set of inference rules. We consider an *ordering-based* inference system, that assumes an ordering $\succ$ on terms and literals, and uses it to restrict expansion inferences and define contraction inferences. This ordering is a *complete simplification ordering* (stable, monotone, with the subterm property, hence well-founded, and total on ground terms and literals). An *inference rule* γ with n premises is an $n+1$-ary relation on clauses. Each inference rule has a *main premise* that yields the conclusion in the context of the other (*side*) premises. For contraction rules, the main premise is reduced to the conclusion. Let I be a mapping, called a *model functor*, that assigns to each set of ground clauses N not containing $\square$ an interpretation I_N, called the *candidate model.* An inference system $\mathit{\Gamma}$ has the *reduction property for counterexamples*, if for all sets N of clauses and minimal counterexamples C for I_N in N, there is an inference in Γ from N with main premise C, side premises that are true in I_N, and conclusion D that is a smaller counterexample for I_N than C.

3 Variable inactivity in DPLL($\Gamma + \mathcal{T}$)

In this section we will see how previous results from the rewrite-based approach to satisfiability procedures [1, 8] can be imported into the DPLL($\Gamma + \mathcal{T}$) framework to combine a built-in theory $\mathcal{T}$ and an axiomatized theory $\mathcal{R}$. In a purely rewrite-based approach there is no built-in theory and all axioms are part of the input in $\mathcal{R}$. The core of the methodology is to show that a first-order engine, such as SP, is an $\mathcal{R}$-satisfiability procedure, by showing that it is guaranteed to terminate on $\mathcal{R}$-satisfiability problems $\mathcal{R} \uplus S$, where S is a set of ground unit $\mathcal{R}$-clauses. Termination is *modular*: if SP terminates on $\mathcal{R}_i$-satisfiability problems, for $1 \leq i \leq n$, it terminates also on $\mathcal{R}$-satisfiability problems for $\mathcal{R} = \bigcup_{i=1}^{n} \mathcal{R}_i$, provided the signatures of the $\mathcal{R}_i$'s do not share function symbols, and all the $\mathcal{R}_i$'s are *variable inactive* [1]:

Definition 1. *A clause C is* variable-inactive *if no maximal literal in C is an equation $t \simeq x$ where $x \notin Var(t)$. A set of clauses is* variable-inactive *if all its clauses are.*

Maximality is relative to the ordering $\succ$ of Γ, which is required to be *good*, meaning that $t \succ c$ for all ground compound term t and constant c [1, 6].

Definition 2. *A theory presentation $\mathcal{R}$ is* variable-inactive *for an inference system Γ if the limit S_∞ of a fair Γ-derivation from $S_0 = \mathcal{R} \uplus S$ is variable-inactive, where S is a set of ground unit $\mathcal{R}$-clauses.*

It was proved in [1] (cf. Thm. 4.5) that if $\mathcal{R}$ is variable-inactive, then it is stably-infinite. This observation is a corollary of a result of [8] (cf. Lemma 5.2) that says that if S_0 is satisfiable, then S_0 admits no infinite models if and only if the limit S_∞ of a fair SP-derivation from S_0 contains a *cardinality constraint*, that is, a clause containing only non-trivial (i.e., other than $x \simeq x$) positive equations between variables (e.g., $y \simeq x \vee y \simeq z$). Such a clause is clearly not variable-inactive. SP will reveal the lack of stable infiniteness by generating a cardinality constraint.[5] Thus, variable-inactivity is a sufficient condition for modularity of termination, hence to combine theories in the rewrite-based approach, and for stable-infiniteness, hence to mix combination of axiomatized theories as in the rewrite-based approach with combination of built-in theories à la Nelson-Oppen, as investigated also in [7] in a different setting.

[5] Lemma 5.2 in [8] requires that the superposition-based inference system is invariant with respect to renaming finitely many constants. Most inference systems satisfy a stronger requirement, namely they allow signature extensions, e.g., to introduce Skolem constants.

In DPLL($\mathit{\Gamma}$+$\mathcal{T}$) applied to a problem $\mathcal{R} \uplus P$ modulo $\mathcal{T}$, $\mathit{\Gamma}$ deals only with non-ground clauses and ground unit clauses, so that $\mathit{\Gamma}$ works on an $\mathcal{R}$-satisfiability problem $\mathcal{R} \uplus S$, where S is a set of ground unit clauses. Thus, it makes sense to apply the results from the rewrite-based approach to $\mathit{\Gamma}$ seen as an $\mathcal{R}$-solver. DPLL($\mathit{\Gamma}$+$\mathcal{T}$) needs to combine $\mathcal{T}_1, \dots, \mathcal{T}_n, \mathcal{R}$ in the Nelson-Oppen scheme, which requires that the theories do not share function symbols, are stably infinite and each solver generates all entailed (disjunctions of) equalities between constants. We assume that $\mathcal{T}_1, \dots, \mathcal{T}_n$ satisfy these requirements and that $\mathcal{R}$ does not share function symbols with them. For stable infiniteness of $\mathcal{R}$, we apply the above result about variable inactivity implying stable infiniteness: in the new version of Z3(SP), the SP engine is equipped with a test that detects the generation of variable-inactive clauses, hence cardinality constraints, and discovers whether $\mathcal{R}$ is not stably infinite. Such a test also excludes upfront a situation such as $\mathcal{R} = \{x = a \vee x = b\}$ of the example in Section 1. For the generation of (disjunctions of) equalities between constants in $\mathcal{R}$, we assume that the $\mathit{\Gamma}$ engine is *fair*, which ensures that every theorem is implied by some generated formulae.[6] If contraction is also done systematically, only irredundant clauses generated by $\mathit{\Gamma}$ are kept and passed to the DPLL($\mathcal{T}$) core.

The aforementioned results on variable inactivity were proved under the hypotheses that the ground unit clauses in S are $\mathcal{R}$-clauses and equality is the only predicate symbol. In the framework of DPLL($\mathit{\Gamma} + \mathcal{T}$), ground clauses may contain also $\mathcal{T}$-symbols, and $\mathcal{R}$ may introduce predicate symbols other than equality. We handle the first issue by *purification*, a standard step in the Nelson-Oppen method, which separate occurrences of function symbols from different signatures, by introducing new constant symbols (e.g., $f(g(a)) \simeq b$, where f and g belongs to different signatures, becomes $f(c) \simeq b \wedge g(a) \simeq c$, where c is new). The initial set of ground clauses P is transformed in two disjoint sets P_1 and P_2, where P_1 contains only $\mathcal{R}$-symbols and P_2 only $\mathcal{T}$-symbols. Since only constants are introduced, the problem remains ground. We deal with the second issue by representing an $\mathcal{R}$-atom $p(t_1, \dots, t_n)$ as $f_p(t_1, \dots, t_n) = \top$, where f_p is a new function symbols and $\top$ is a special constant.

Definition 3. *A set of formulæ $\mathcal{R} \uplus P$ is* smooth *with respect to a background theory $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$, if the signatures of $\mathcal{T}_1, \dots, \mathcal{T}_n, \mathcal{R}$ do not share function symbols, $\mathcal{R}$ is variable inactive, and P is a set of ground formulæ $P_1 \uplus P_2$, where P_1 contains only $\mathcal{R}$-symbols, and P_2 only $\mathcal{T}$-symbols.*

This definition summarizes the problem requirements for the sequel.

4 Unsound theorem proving in DPLL($\mathit{\Gamma} + \mathcal{T}$)

In theorem proving applied to mathematics, most conjectures are true. Thus, it is customary to sacrifice completeness for efficiency, and retain soundness,

[6] Fairness guarantees that inferences are done systematically, in such a way that every theorem has a minimal proof in the limit: see [4] for details.

which is necessary to attribute unsatisfiability to a set of clauses F if a proof is found. A traditional example is *deletion by weight* [19], where clauses that are too "heavy" are deleted. In theorem proving applied to verification, most conjectures are false. Thus, it was suggested in [17] to sacrifice soundness for termination, and retain completeness, which is necessary to establish satisfiability if a proof is *not* found. Dually to deletion by weight, an unsound inference could suppress literals in clauses that are too heavy.

We consider a single unsound inference rule: adding an arbitrary clause C. This rule is unsound because C may not be implied by F. This rule is simple, but can simulate different kinds of unsound inferences. Suppose we want to suppress the literals D in $C \vee D$, then we can simply add C, which subsumes $C \vee D$. Suppose a clause $C[t]$ contains a deep term t, and we want to replace it with a constant a. We can accomplish this by adding $t \simeq a$. The idea is to extend DPLL($\Gamma + \mathcal{T}$) with a *reversible* unsound inference rule. We say it is *reversible*, because we track the consequences of the clauses added by this rule.

DPLL($\Gamma + \mathcal{T}$) works on *hypothetical clauses* of the form $H \triangleright C$, where C is a clause (i.e., a disjunction of literals), and H is the set of ground literals, from the candidate model built by DPLL($\Gamma + \mathcal{T}$), that C depends on, in the sense that they were used as premises to infer C by Γ-inferences. The set of hypotheses should be interpreted as a conjunction, and a hypothetical clause $(l_1 \wedge \ldots \wedge l_n) \triangleright (l'_1 \vee \ldots l'_m)$ should be interpreted as $\neg l_1 \vee \ldots \vee \neg l_n \vee l'_1 \vee \ldots \vee l'_m$. In this context, rather than merely adding a clause C, the unsound inference rule introduces a hypothetical clause $\lceil C \rceil \triangleright C$, where $\lceil C \rceil$ is a new propositional variable that is used to track the consequences of adding C. Note that the hypothetical clause $\lceil C \rceil \triangleright C$ is semantically equivalent to $\neg \lceil C \rceil \vee C$. This clause does not change the satisfiability of the input formula because $\lceil C \rceil$ is a new propositional variable.

The DPLL($\Gamma + \mathcal{T}$) calculus is described as a transition system [11]. States of the transition system are of the form $M \parallel F$, where M is a sequence of *assigned literals*, and F a set of *hypothetical clauses*. Intuitively, M represents a partial assignment to ground literals, with their justifications, and therefore it represents a partial model, or a set of candidate models. An assigned literal can be either a *decided literal* or an *implied literal*. A decided literal represents a guess, and an implied literal l_C a literal l that was implied by a clause C. No assigned literal occurs twice in M nor does it occur negated in M. If neither l nor $\neg l$ appears in M, then l is said to be *undefined*. The initial state is $\parallel F_0$, where F_0 is the set $\{\emptyset \triangleright C \mid C \in \mathcal{R} \uplus P\}$. During conflict resolution, we also use states of the form $M \parallel F \parallel C$, where C is a ground clause. In the following, *clauses*(F) denotes the set $\{C \mid H \triangleright C \in F\}$, $M \models_P C$ indicates that M propositionally satisfies C, and if C is the clause $l_1 \vee \ldots \vee l_n$, then $\neg C$ is the formula $\neg l_1 \wedge \ldots \wedge \neg l_n$. We use *lits*($M$) to denote the set of assigned literals, *ngclauses*(F) for the subset of non-ground clauses of *clauses*(F), and *clauses**($M \parallel F$) for *ngclauses*(F) $\cup$ *lits*(M). We also write C instead of $\emptyset \triangleright C$.

We extend the calculus with the rule UnsoundIntro. This rule introduces an arbitrary clause C into F, and it adds the ground literal $\lceil C \rceil$ to M, where $\lceil C \rceil$

is a new propositional variable used as a label for clause C. The idea is to record the fact that we are *guessing* C.

UnsoundIntro

$$M \,\|\, F \quad\quad \Longrightarrow M \; \lceil C \rceil \,\|\, F,\; \lceil C \rceil \triangleright C \quad\quad \textbf{if} \begin{cases} C \notin \mathit{clauses}(F), \\ \lceil C \rceil \; \mathit{is\ new}, \\ \lceil C \rceil, \neg \lceil C \rceil \notin M, \end{cases}$$

where the side condition prevents the system from adding C, if it is already known to be inconsistent with the partial model M.

In order to combine the theories in $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$ and $\mathcal{R}$ in the Nelson-Oppen scheme, we need to communicate to $\mathcal{R}$ the (disjunctions of) equalities between constants entailed by $\mathcal{T}$ and P. The next inference rule takes care of this requirement, which we relax as in [12], because the $\mathcal{T}_i$-solvers for linear arithmetic and bit-vectors can build a specific candidate $\mathcal{T}$-model for M, that we denote by $\mathsf{model}(M)$. The idea is to inspect $\mathsf{model}(M)$ and propagate all the equalities it implies, hedging that they are consistent with $\mathcal{R}$. Since these equalities are *guesses*, if one of them is inconsistent with $\mathcal{R}$, backtracking will be used to fix $\mathsf{model}(M)$. The rationale for this approach is practical: it is generally far less expensive to enumerate the equalities satisfied in a particular $\mathcal{T}$-model than those satisfied by all $\mathcal{T}$-models consistent with M; the number of equalities that really matter is small in practice.

PropagateEq

$$M \,\|\, F \quad\quad \Longrightarrow M \; t \simeq s \,\|\, F \quad \textbf{if} \begin{cases} t \text{ and } s \text{ are ground}, \\ t, s \text{ occur in } F, \\ (t \simeq s) \;\text{ is undefined in } M, \\ \mathsf{model}(M)(t) = \mathsf{model}(M)(s). \end{cases}$$

The basic and theory propagation rules of DPLL($\Gamma + \mathcal{T}$) are repeated from [11] in Figure 1.

The interface with the inference system Γ is realized by the Deduce rule: assume γ is an inference rule of Γ with n premises, $\{H_1 \triangleright C_1, \ldots, H_m \triangleright C_m\}$ is a set of hypothetical clauses in F, $\{l_{m+1}, \ldots, l_n\}$ is a set of assigned literals in M, and $\mathsf{H}(\gamma)$ denotes the set $H_1 \cup \ldots \cup H_m \cup \{l_{m+1}, \ldots, l_n\}$; then γ is applied to the set of premises $\mathsf{P}(\gamma) = \{C_1, \ldots, C_m, l_{m+1}, \ldots, l_n\}$, and the conclusion $\mathsf{C}(\gamma)$ is added to F as $\mathsf{H}(\gamma) \triangleright \mathsf{C}(\gamma)$. The hypotheses of the clauses $H_i \triangleright C_i$ are hidden from the inference rules in Γ. Our Deduce rule is slightly different from its predecessor, named Deduce$^\sharp$ in [11]: Deduce$^\sharp$ allowed Γ to use as premises non-ground clauses and ground unit clauses in *clauses*(F), whereas our Deduce allows it to use only non-ground clauses in *clauses*(F). This is a consequence of the addition of PropagateEq, which adds the relevant ground unit clauses directly to M, so that Γ finds them in *lits*(M). This is also the reason why we let PropagateEq add equalities between ground terms and not only between constants.

We say a hypothetical clause $H \triangleright C$ is in *conflict* if every literal in C is complementary to an assigned literal. The Conflict rule converts a hypothetical

Decide

$$M \parallel F \Longrightarrow M\ l \parallel F \qquad \textbf{if} \begin{cases} l \text{ is ground,} \\ l \text{ or } \neg l \text{ occurs in } F, \\ l \text{ is undefined in } M. \end{cases}$$

UnitPropagate

$$M \parallel F, H \rhd (C \vee l) \Longrightarrow M\ l_{H\rhd(C\vee l)} \parallel F, H\rhd(C\vee l) \qquad \textbf{if} \begin{cases} l \text{ is ground,}\\ M \models_P \neg C,\\ l \text{ is undefined in } M.\end{cases}$$

Deduce

$$M\parallel F \Longrightarrow M\parallel F, \mathsf{H}(\gamma)\rhd\mathsf{C}(\gamma) \qquad \textbf{if} \begin{cases}\gamma\in \mathit{\Gamma},\\ \mathsf{P}(\gamma)\subseteq \mathit{clauses}^\star(M\parallel F),\\ \mathsf{C}(\gamma)\notin \mathit{clauses}(F).\end{cases}$$

Conflict

$$M\parallel F, H\rhd C \Longrightarrow M\parallel F,H\rhd C\parallel \neg H\vee C \qquad \textbf{if}\ M\models_P \neg C$$

Explain

$$M\parallel F\parallel C\vee\bar{l} \Longrightarrow M\parallel F\parallel \neg H\vee D\vee C \qquad \textbf{if}\ l_{H\rhd(D\vee l)}\in M$$

Learn

$$M\parallel F\parallel C \Longrightarrow M\parallel F, C\parallel C \qquad \textbf{if}\ C\notin \mathit{clauses}(F)$$

Backjump

$$M\ l'\ M'\parallel F\parallel C\vee l \Longrightarrow M\ l_{C\vee l}\parallel F' \qquad \textbf{if}\begin{cases} M\models_P\neg C,\\ l \text{ is undefined in } M,\\ F' = \left\{\begin{array}{l} H\rhd C\in F \mid \\ \qquad H\cap \mathit{lits}(l'\ M')=\emptyset\end{array}\right\}\end{cases}$$

Unsat

$$M\parallel F\parallel \square \Longrightarrow \mathit{unsat}$$

T-Propagate

$$M\parallel F \Longrightarrow M\ l_{(\neg l_1\vee\ldots\vee\neg l_n\vee l)}\parallel F \qquad \textbf{if}\begin{cases} l \text{ is ground and occurs in } F,\\ l \text{ is undefined in } M,\\ l_1,\ldots,l_n\in \mathit{lits}(M),\\ l_1,\ldots,l_n\models_T l.\end{cases}$$

T-Conflict

$$M\parallel F \Longrightarrow M\parallel F\parallel \neg l_1\vee\ldots\vee\neg l_n \qquad \textbf{if}\begin{cases} l_1,\ldots,l_n\in \mathit{lits}(M),\\ l_1,\ldots,l_n\models_T \mathit{false}.\end{cases}$$

Fig. 1. Basic and theory propagation rules

conflict clause $H \triangleright C$ into a regular clause by negating its hypotheses, and puts the DPLL($\mathit{\Gamma} + \mathcal{T}$) system in conflict resolution mode. The Explain rule unfolds literals from conflict clauses that were produced by unit propagation. Any clause derived by Explain can be added to F by the Learn rule, because it is a logical consequence of the original set of clauses. The rule Backjump drives the DPLL($\mathit{\Gamma} + \mathcal{T}$) system back from conflict resolution to search mode, and it unassigns at least one decided literal (l' in the rule definition). All hypothetical clauses $H \triangleright C$ which contain hypotheses that will be unassigned by the Backjump rule are deleted. Note that a learnt clause D may contain $\neg\lceil C \rceil$. In this case, the clause D is recording the context where guessing the clause C is unsound.

It was proved in [11] that DPLL($\mathit{\Gamma} + \mathcal{T}$) is refutationally complete when $\mathcal{T}$ is empty. We prove a stronger result for the case where $\mathcal{T}$ is not empty. We say a state $M \,\|\, F$ is *saturated* if the only applicable rule is UnsoundIntro.

Theorem 1. *If the initial set of clauses $S = \mathcal{R} \uplus P$ is smooth, and $\mathit{\Gamma}$ has the reduction property for counterexamples, whenever $M \,\|\, F$ is saturated, S is satisfiable modulo the background theory $\mathcal{T}$.*

All inference systems considered in the rest of this paper satisfy the reduction property for counterexamples.

We assign an *inference depth* to every clause in *clauses*(F) and literal in *lits*(M). Intuitively, the inference depth of a clause C indicates the depth of the derivation needed to produce C. More precisely, all clauses in the original set of clauses have inference depth 0. If a clause C is produced using the Deduce rule, and n is the maximum inference depth of the premises, then the inference depth of C is $n + 1$. The inference depth of a literal l_C in M is equal to the inference depth of C. If l is a decided literal, and n is the minimum inference depth of the clauses in F that contain l, then the inference depth of l is n. We say DPLL($\mathit{\Gamma} + \mathcal{T}$) is $\langle k_d, k_u \rangle$-bounded if Deduce is restricted to premises with inference depth $< k_d$, and UnsoundIntro can only be applied k_u times.

Theorem 2. *$\langle k_d, k_u \rangle$-bounded DPLL($\mathit{\Gamma} + \mathcal{T}$) always terminates.*

A state $M \,\|\, F$ is *stuck* at k_d if the only applicable rules are UnsoundIntro and Deduce, and Deduce is only applicable to premises with inference depth $\geq k_d$. Theorem 2 suggests a simple saturation strategy where the bounds k_d and k_u are increased whenever the procedure reaches a blocked state.

We use U to denote a sequence of "unsound axioms" introduced by Unsound-Intro. In the next section, we investigate some examples where DPLL($\mathit{\Gamma} + \mathcal{T}$) is a decision procedure for a smooth set of formulæ S. This is accomplished by showing that for some sequence of "unsound axioms" U, there are k_d and k_u, such that $\langle k_d, k_u \rangle$-bounded DPLL($\mathit{\Gamma} + \mathcal{T}$) is guaranteed to terminate in the *unsat* state, whenever S is unsatisfiable, and in a state $M \,\|\, F$ which is not *stuck* at k_d, whenever S is satisfiable.

Due to space limitations, we refer to [11] for the contraction inference rules of DPLL($\mathit{\Gamma} + \mathcal{T}$). DPLL($\mathit{\Gamma} + \mathcal{T}$) assigns a *scope level* to each literal in M. The scope level of a literal l, *level*(l), in $M \; l \; M'$, is equal to the number of decided literals in $M\ l$. The level of a set of literals H is $\mathit{level}(H) = \mathit{max}\{\mathit{level}(l) \mid l \in H\}$. A contraction rule γ from Γ is generalized to hypothetical clauses as follows: given a main premise $H \triangleright C$, and side premises $H_2 \triangleright C_2, \ldots, H_m \triangleright C_m$, and $l_{m+1}, \ldots, l_n$, taken from F and $\mathit{lits}(M)$, respectively, let $H' = H_2 \cup \ldots \cup H_m \cup \{l_{m+1}, \ldots, l_n\}$. Assume that γ applies to the premises $C, C_2, \ldots, C_m, l_{m+1}, \ldots, l_n$. If $\mathit{level}(H) \geq \mathit{level}(H')$, we claim it is safe to delete $H \triangleright C$. In contrast, if $\mathit{level}(H) < \mathit{level}(H')$, then it is only safe to *disable* the clause $H \triangleright C$ until $\mathit{level}(H')$ is backjumped. A *disabled* clause is not deleted, but it is not used as premise until it is re-enabled.

Example 1. Let $\mathcal{R}$ be $\{\neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq z) \vee x \sqsubseteq z,\ \neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y)\}$, and P be $\{a \sqsubseteq b,\ a \sqsubseteq f(c),\ \neg(a \sqsubseteq c)\}$. Assume Γ features Resolution, Superposition and Simplification. If UnsoundIntro adds $\lceil f(x) \simeq x \rceil \triangleright f(x) \simeq x$, the monotonicity axiom and $a \sqsubseteq f(c)$ get rewritten. Note that $\lceil f(x) \simeq x \rceil$ is a decision literal, and $\mathit{level}(\lceil f(x) \simeq x \rceil) = 1$. Thus, the rewriting step only disables $a \sqsubseteq f(c)$, and adds $\lceil f(x) \simeq x \rceil \triangleright a \sqsubseteq c$ to F. Resolution generates the conflict clause $\lceil f(x) \simeq x \rceil \triangleright \Box$. Using the conflict resolution rules, the literal $\neg\lceil f(x) \simeq x \rceil$ is added to M, preventing DPLL($\Gamma + \mathcal{T}$) from guessing $f(x) \simeq x$ again. Next, if UnsoundIntro adds $\lceil f(f(x)) \simeq x \rceil \triangleright f(f(x)) \simeq x$, monotonicity and $a \sqsubseteq b$ produce only $f(a) \sqsubseteq f(b)$, while monotonicity and $a \sqsubseteq f(c)$ produce only $f(a) \sqsubseteq f(f(c))$, which is disabled and replaced by $\lceil f(f(x)) = x \rceil \triangleright f(a) \sqsubseteq c$. Then, DPLL($\Gamma + \mathcal{T}$) reaches a saturated state, and satisfiability is detected.

Example 2. Let $\mathcal{R}$ be $\{\neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq z) \vee x \sqsubseteq z\}$, P be $\{a \sqsubseteq b_1,\ b_2 \sqsubseteq c,\ \neg(a \sqsubseteq c), b_1 \leq b_2, b_1 > b_2 - 1\}$, and $\mathcal{T}$ be the theory of linear integer arithmetic. Assume Γ is Hyperresolution, Superposition and Simplification. UnitPropagate adds the literals of P to M. In the model $\mathsf{model}(M)$ maintained by the linear arithmetic solver, $\mathsf{model}(M)(b_1) = \mathsf{model}(M)(b_2)$. Thus, PropagateEq *guesses* the equation $b_1 \simeq b_2$. Say $b_2 \succ b_1$: Simplification rewrites $b_2 \sqsubseteq c$ to $b_1 \sqsubseteq c$. Hyperresolution derives $a \sqsubseteq c$ from $a \sqsubseteq b_1$, $b_1 \sqsubseteq c$ and transitivity, so that an inconsistency is detected. DPLL($\Gamma + \mathcal{T}$) backtracks and adds $\neg(b_1 \simeq b_2)$ to M. T-Conflict detects the inconsistency between this literal and $\{b_1 \leq b_2, b_1 > b_2 - 1\}$. The conflict resolution rules are applied again and the empty clause is produced.

5 Essentially finite theories

We say a structure Φ is *essentially finite* with respect to the function symbol f if $\Phi(f)$ has finite range. Essential finiteness is slightly weaker than finiteness, because it admits an infinite domain provided the range of $\Phi(f)$ is finite.

Theorem 3. *If Φ is an essentially finite structure with respect to a monadic function symbol f, then there exist k_1, k_2, $k_1 \neq k_2$, such that $\Phi \models f^{k_1}(x) \simeq f^{k_2}(x)$.*

Proof. For all $v \in |\Phi|$, we call *f-chain* starting at v, the sequence:

$$v = \Phi(f)^0(v),\ \Phi(f)^1(v),\ \Phi(f)^2(v), \ldots,\ \Phi(f)^i(v), \ldots$$

Since $\Phi(f)$ has finite range, there exist q_1, q_2, with $q_1 \neq q_2$, such that $\Phi(f)^{q_1}(v) = \Phi(f)^{q_2}(v)$. Say that $q_1 > q_2$. Then we call *size*, denoted $\mathsf{sz}(\Phi, f, v)$, and *prefix*, denoted $\mathsf{pr}(\Phi, f, v)$, of the f-chain starting at v, the smallest q_1 and q_2, respectively, such that $\Phi(f)^{q_1}(v) = \Phi(f)^{q_2}(v)$ and $q_1 > q_2$. We term *lasso*, denoted $\mathsf{ls}(\Phi, f, v)$, of the f-chain starting at v, the difference between size and prefix, that is, $\mathsf{ls}(\Phi, f, v) = \mathsf{sz}(\Phi, f, v) - \mathsf{pr}(\Phi, f, v)$. We say that $\Phi(f)^n(v)$ is *in the lasso* of the f-chain starting at v, if $n \geq \mathsf{pr}(\Phi, f, v)$. Clearly, for all elements u in the lasso of the f-chain starting at v, $\Phi(f)^m(u) = u$, when $m = \mathsf{ls}(\Phi, f, v)$. Also, for all multiples of the lasso, that is, for all $l = h \cdot \mathsf{ls}(\Phi, f, v)$ for some $h > 0$, $\Phi(f)^l(u) = u$. Let $p = \mathsf{max}\{\mathsf{pr}(\Phi, f, v) \mid v \in \mathsf{range}(\Phi(f))\} + 1$ and $l = \mathsf{lcm}\{\mathsf{ls}(\Phi, f, v) \mid v \in \mathsf{range}(\Phi(f))\}$, where lcm abbreviates least common multiple. We claim that $\Phi \models f^{p+l}(x) \simeq f^p(x)$, that is, $k_1 = p + l$ and $k_2 = p$. By way of contradiction, assume that for some $v \in |\Phi|$, $\Phi(f)^{p+l}(v) \neq \Phi(f)^p(v)$. Take the f-chain starting at v: $\Phi(f)^p(v)$ is in the lasso of this chain, because $p \geq \mathsf{pr}(\Phi, f, v)$. Since l is a multiple of $\mathsf{ls}(\Phi, f, v)$, we have $\Phi(f)^{p+l}(v) = \Phi(f)^l(\Phi(f)^p(v)) = \Phi(f)^p(v)$, a contradiction. $\square$

Example 3. Let Φ be a structure such that $|\Phi| = \{v_0, v_1, v_2, \ldots, v_9, \ldots\}$, and let $\Phi(f)$ be the function defined by the following mapping: $\{v_0 \mapsto v_1, v_1 \mapsto v_2, v_2 \mapsto v_3, v_3 \mapsto v_4, v_4 \mapsto v_2, v_5 \mapsto v_6, v_6 \mapsto v_7, v_7 \mapsto v_8, v_8 \mapsto v_5, * \mapsto v_9\}$, where $*$ stands for any other element. The f-chain starting at v_0 has $\mathsf{pr}(\Phi, f, v_0) = 2$, $\mathsf{sz}(\Phi, f, v_0) = 5$ and $\mathsf{ls}(\Phi, f, v_0) = 3$. The f-chain starting at v_5 has $\mathsf{pr}(\Phi, f, v_5) = 0$, $\mathsf{sz}(\Phi, f, v_5) = 4$ and $\mathsf{ls}(\Phi, f, v_5) = 4$. Then, $p = 2+1 = 3$, $l = 12$, $k_1 = p+l = 15$ and $k_2 = p = 3$, and $\Phi \models f^{15}(x) \simeq f^3(x)$.

To identify classes of problems for which DPLL($\Gamma + \mathcal{T}$) is a decision procedure, we focus on theories $\mathcal{R}$ that satisfy either one of the following properties:

Definition 4. *$\mathcal{R}$ has the* finite model property*, if for all sets P of ground $\mathcal{R}$-clauses, such that $\mathcal{R} \uplus P$ is satisfiable, $\mathcal{R} \uplus P$ has a model Φ with finite $|\Phi|$.*

Definition 5. *Let $\mathcal{R}$ be a presentation whose signature contains a single monadic function symbol f. $\mathcal{R}$ is* essentially finite*, if for all sets P of ground $\mathcal{R}$-clauses, such that $\mathcal{R} \uplus P$ is satisfiable, $\mathcal{R} \uplus P$ has a model Φ, such that* $\mathsf{range}(\Phi(f))$ *is finite.*

We show that essentially finite theories can give rise to decision procedures if clause length is bounded.

Theorem 4. *Let $\mathcal{R}$ be an essentially finite theory and P a set of ground clauses. Let Γ be a rewrite-based inference system. Consider a DPLL($\Gamma + \mathcal{T}$) procedure where* $\mathsf{UnsoundIntro}$ *progressively adds all equations of the form $f^j(x) \simeq f^k(x)$ with $j > k$. Then DPLL($\Gamma + \mathcal{T}$) is a decision procedure for the satisfiability modulo $\mathcal{T}$ of smooth problems in the form $\mathcal{R} \uplus P$ if there exists an n such that no clause created contains more than n literals.*

Proof. If $\mathcal{R} \uplus P$ is unsatisfiable, then by completeness DPLL($\Gamma + \mathcal{T}$) will generate the empty clause when k_d becomes large enough. If $\mathcal{R} \uplus P$ is satisfiable, choose

k_u large enough to contain the axiom $f^{k_1}(x) \simeq f^{k_2}(x)$ as given in Theorem 3. We need to prove that if k_d is large enough, DPLL($\Gamma + \mathcal{T}$) will not get stuck at k_d. To do that, we prove that only a finite number of clauses are generated for unbounded k_d for the given k_u. The axiom $f^{k_1}(x) \simeq f^{k_2}(x)$ is oriented into the rewrite rule $f^{k_1}(x) \to f^{k_2}(x)$. This guarantees that no term $f^k(t)$ with $k > k_1$ is kept. Since no clause can contain more than n literals, only a finite number of clauses can be derived for an unbounded k_d. $\square$

Assume that Γ is Superposition with negative selection plus Hyperresolution[7]. If $\mathcal{R}$ is Horn, Superposition is Unit Superposition, which does not increase clause length, and Hyperresolution only generates positive unit clauses, so that no clause containing more than n literals can be produced. If $\mathcal{R}$ is a set of nonequality clauses with no more than two literals each, and Γ is Resolution plus Simplification (to apply $f^{k_1}(x) \to f^{k_2}(x)$), then all generated clauses contain at most two literals. To give further examples, we need the following:

Definition 6. *A clause* $C = \neg l_1 \vee \ldots \vee \neg l_n \vee l_{n+1} \vee \ldots \vee l_{n+m}$ *is* ground-preserving *if* $\bigcup_{j=n+1}^{n+m} Var(l_j) \subseteq \bigcup_{j=1}^{n} Var(l_j)$. *A set is* ground-preserving *if all its clauses are.*

In a ground-preserving[8] set the only positive clauses are ground. If $\mathcal{R}$ is ground-preserving, Hyperresolution only generates ground clauses; Superposition with negative selection yields either ground clauses or ground-preserving clauses with decreasing number of variable positions, so that no new non-ground terms can be created, and only finitely many non-ground ground-preserving clauses can be derived. If $\mathcal{R}$ is also essentially finite, the depth of terms is limited by Simplification by $f^{k_1}(x) \to f^{k_2}(x)$, so that only finitely many ground clauses can be generated. Below, we show that some specific theories relevant to the axiomatization of type systems in programming languages are essentially finite and satisfy the properties of Theorem 4. Given the axioms

$$\begin{aligned}
\textsf{Reflexivity} \quad & x \sqsubseteq x && (1)\\
\textsf{Transitivity} \quad & \neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq z) \vee x \sqsubseteq z && (2)\\
\textsf{Anti-Symmetry} \quad & \neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq x) \vee x \simeq y && (3)\\
\textsf{Monotonicity} \quad & \neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y) && (4)\\
\textsf{Tree-Property} \quad & \neg(z \sqsubseteq x) \vee \neg(z \sqsubseteq y) \vee x \sqsubseteq y \vee y \sqsubseteq x && (5)
\end{aligned}$$

$\textsf{MI} = \{(1),(2),(3),(4)\}$ presents a type system with *multiple inheritance*, and $\textsf{SI} = \textsf{MI} \uplus \{(5)\}$ presents a type system with *single inheritance*, where $\sqsubseteq$ is the subtype relationship and f is a type constructor.

Theorem 5. SI *has the finite model property hence it is essentially finite.*

[7] Hyperresolution is realized by Resolution with negative selection rule.

[8] This notion is a weakening of that of "positive variable dominated" clause of Definition 3.18 in [9].

Proof. Assume $\mathsf{SI} \uplus P$ is satisfiable, and let Φ be a model for it. It is sufficient to show there is a finite model Φ'. Let T_P be the set of subterms of terms in P, and V_P be the set $\Phi(T_P)$. Since P is finite and ground, V_P is finite. Let $|\Phi'|$ be $V_P \cup \{r\}$, where r is an element not in V_P. Then, we define $\Phi'(\sqsubseteq)(v_1, v_2)$ as:

$$r = v_2 \text{ or } (v_1, v_2) \in \Phi(\sqsubseteq).$$

Intuitively, r is a new maximal element. $\langle |\Phi'|, \Phi'(\sqsubseteq)\rangle$ is a poset and $\Phi'(\sqsubseteq)$ satisfies the $\mathsf{Tree\text{-}Property}$. Now, we define an auxiliary function $g\colon |\Phi'| \to |\Phi'|$ as:

$$g(v) = \begin{cases} \Phi(f)(v) \text{ if } f(t) \in T_P, \text{ and } \Phi(t) = v; \\ r \qquad\quad\; \text{otherwise.} \end{cases}$$

Let dom_f, the relevant domain of f, be the set $\{\Phi(t) \mid f(t) \in T_P\} \cup \{r\}$. With a small abuse of notation, we use $v \sqsubseteq w$ to denote $(v, w) \in \Phi'(\sqsubseteq)$. Then, we define $\Phi'(f)(v)$ as $g(w)$, where w is an element in $|\Phi'|$ such that $v \sqsubseteq w$, $w \in \mathsf{dom}_f$, and for all w', $v \sqsubseteq w'$ and $w' \in \mathsf{dom}_f$ imply $w \sqsubseteq w'$. This function is well defined because $\Phi'(\sqsubseteq)$ satisfies the $\mathsf{Tree\text{-}Property}$, r is the maximal element of $|\Phi'|$, and $r \in \mathrm{dom}_f$. Moreover, $\Phi'(f)$ is monotonic with respect to $\Phi'(\sqsubseteq)$. □

Definition 7. *Let $\langle A, \sqsubseteq\rangle$ be a poset. The* Dedekind-MacNeille completion [18] *of $\langle A, \sqsubseteq\rangle$ is the unique complete lattice $\langle B, \preceq\rangle$ satisfying the following properties.*

– *There is an injection α from A to B such that: $v_1 \sqsubseteq v_2$ iff $\alpha(v_1) \preceq \alpha(v_2)$,*
– *Every subset of B has a greatest (least) lower bound, and*
– *B is finite if A is finite. Actually, B is a subset of 2^A.*

Theorem 6. MI *has the finite model property hence it is essentially finite.*

Proof. The construction used for SI does not work for MI, because without the $\mathsf{Tree\text{-}Property}$ the w in the definition of $\Phi'(f)(v)$ may not be unique for a given v. First, we define an auxiliary structure Φ_0 such that $|\Phi_0| = V_P$, $\Phi_0(\sqsubseteq) = \Phi(\sqsubseteq)|_{V_P}$, and $\Phi_0(f)$ is defined as:

$$\Phi_0(f)(v) = \begin{cases} \Phi(f)(v) \text{ if } f(t) \in T_P, \text{ and } \Phi(t) = v, \\ w \qquad\quad\; \text{otherwise,} \end{cases}$$

where w is some element of V_P. Note that $\langle V_P, \Phi_0(\sqsubseteq)\rangle$ is a poset. Let dom_f be the set $\{\Phi(t) \mid f(t) \in T_P\}$. Then, following [10] we use the Dedekind-MacNeille completion to complete $\langle V_P, \Phi_0(\sqsubseteq)\rangle$ into a complete lattice $\langle B, \preceq\rangle$. We use $glb(S)$ to denote the greatest lower bound of a subset S of B. Now, we define a finite model Φ' for $\mathsf{MI} \uplus P$ with domain $|\Phi'| = B$, in the following way:

$$\begin{aligned} \Phi'(c) &= \alpha(\Phi_0(c)) \quad \text{for every constant } c \text{ in } T_P, \\ \Phi'(\sqsubseteq) &= \preceq, \\ \Phi'(f)(v) &= glb(\{\alpha(\Phi_0(f)(w)) \mid w \in V_P,\ w \in \mathsf{dom}_f,\ v \preceq \alpha(w)\}). \end{aligned}$$

The function $\Phi'(f)$ is monotonic with respect to $\Phi'(\sqsubseteq)$. The structure Φ' satisfies P because for every term t in T_P, we have $\Phi'(t) = \alpha(\Phi(t))$. Moreover, the $\sqsubseteq$-literals in P are satisfied because the lattice $\langle B, \preceq\rangle$ is a Dedekind-MacNeille completion of Φ_0 which is a restriction of Φ. □

Now we show that DPLL($\Gamma + \mathcal{T}$) with the $\mathsf{UnsoundIntro}$ rule is a decision procedure for MI and SI.

Theorem 7. *Let P be a set of ground clauses. Let Γ be Hyperresolution plus Superposition and Simplification. Consider a DPLL($\Gamma + \mathcal{T}$) procedure where UnsoundIntro progressively adds all equations of the form $f^j(x) \simeq f^k(x)$ with $j > k$. Then DPLL($\Gamma + \mathcal{T}$) is a decision procedure for the satisfiability modulo $\mathcal{T}$ of smooth problems in the form $\mathsf{MI} \uplus P$.*

Proof. Since MI is essentially finite, we only need to show that we never generate a clause with more than n literals. This follows from the fact that MI is a Horn theory. $\square$

Theorem 8. *Let P be a set of ground clauses. Let Γ be Hyperresolution plus Superposition and Simplification. Consider a DPLL($\Gamma + \mathcal{T}$) procedure where UnsoundIntro progressively adds all equations of the form $f^j(x) \simeq f^k(x)$ with $j > k$. Then DPLL($\Gamma + \mathcal{T}$) is a decision procedure for the satisfiability modulo $\mathcal{T}$ of smooth problems in the form $\mathsf{SI} \uplus P$.*

Proof. Since SI is essentially finite, we need to show that only finitely many clauses can be generated. SI is not Horn, because of Tree-Property, and it is not ground-preserving, because of Reflexivity. Since all the axioms besides Reflexivity are ground-preserving, any inference will yield either a ground clause or a non-ground ground-preserving clause with fewer variable positions. We just need to consider a Hyperresolution which includes Reflexivity. All those inferences either yield a tautology, a subsumed clause, or a ground clause. $\square$

In Spec# [3], the axiomatization of the type system also contains the axioms $\mathsf{TR} = \{\neg(g(x) \simeq \mathit{null}),\ h(g(x)) \simeq x\}$, where the function g represents the *type representative* of some type. The first axiom states that the representative is never the constant *null*, and the second states that g has an inverse, hence it is injective. Note that any model of TR must be infinite.

Theorem 9. *Let P be a set of ground clauses. Let Γ be Hyperresolution plus Superposition and Simplification. Consider a DPLL($\Gamma + \mathcal{T}$) procedure where UnsoundIntro progressively adds all equations of the form $f^j(x) \simeq f^k(x)$ with $j > k$. Then DPLL($\Gamma + \mathcal{T}$) is a decision procedure for the satisfiability modulo $\mathcal{T}$ of smooth problems in the form $\mathsf{MI} \uplus \mathsf{TR} \uplus P$ and $\mathsf{SI} \uplus \mathsf{TR} \uplus P$.*

Proof. Γ applied to TR and ground equations only generates ground equations of non-increasing depth, hence it terminates. Since MI (SI) and TR do not share function symbols and are variable inactive, Γ terminates also on their union. $\square$

6 Discussion

The DPLL($\Gamma + \mathcal{T}$) system integrates DPLL($\mathcal{T}$) with a generic first-order engine Γ to combine the strengths of DPLL, efficient solvers for special theories such as linear arithmetic, and first-order reasoning based on superposition and resolution. The study in [6] was concerned with using the first-order engine alone as decision procedure, without integrating an SMT-solver. In the method of [7], the first-order engine is used as a pre-processor to compile the theory $\mathcal{R}$ and reduce it to a theory that DPLL($\mathcal{T}$) alone can handle. Thus, it is a two-stage approach. In DPLL($\Gamma+\mathcal{T}$) the first-order engine is tightly integrated within DPLL($\mathcal{T}$). The downside of such a tight integration was that refutational completeness had not been established, except in the case where the background theory $\mathcal{T}$ is empty. In this paper we advanced the DPLL($\Gamma+\mathcal{T}$) approach by giving conditions under which it is refutationally complete when $\mathcal{T}$ is not empty.

Then, we introduced a new calculus that combines DPLL($\Gamma+\mathcal{T}$) with unsound theorem proving. The purpose is to try to enforce termination by introducing additional axioms as hypothesis. A framework for unsound theorem proving was originally given in [17] along with some examples. In the current paper we have provided a mechanism for the prover to detect any unsoundness introduced by the added axioms and recover from it, and we have instantiated the framework with concrete examples for which unsound theorem proving becomes a decision procedure. Some of these examples include monotonicity axioms. Another approach to handle such axioms is *locality*: for instance, extending a theory with a monotonicity axiom is a *local extension* [22, 16]. However, in the applications that motivate our research, there is no guarantee that all relevant instances of $\mathcal{T} \uplus \mathcal{R} \uplus P$ can be seen as hierarchies of local extensions.

Directions for future work include extensions to more presentations, including, for instance, cases where the signature of $\mathcal{R}$ features also non-monadic function symbols (e.g., to cover axioms such as $y \sqsubseteq x \wedge u \sqsubseteq v \Rightarrow \mathit{map}(x,u) \sqsubseteq \mathit{map}(y,v)$). Another open issue is some duplication of reasoning on ground unit clauses in DPLL($\Gamma+\mathcal{T}$), due to the fact that ground unit clauses are seen by both Γ and the congruence closure (CC) algorithm within DPLL($\mathcal{T}$). Using the CC algorithm to compute the completion of the set of ground equations [14, 21], and pass the resulting canonical system to Γ, would not solve the problem entirely, because this solution is not *incremental*, as the addition of a single ground equation requires recomputing the canonical system.

The class of formulæ that can be decided using DPLL($\Gamma+\mathcal{T}$) with unsound inferences includes axiomatizations of type systems, used in tools such as ESC/Java [13] and Spec# [3], which is significant evidence of the relevance of this work to applications.

Acknowledgments Part of this work initiated during a visit of the first author with the Software Reliability Group of Microsoft Research in Redmond.

References

1. A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. New results on rewrite-based satisfiability procedures. *ACM TOCL*, 10(1):129–179, 2009.

2. A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Inf. Comput.*, 183(2):140–164, 2003.
3. M. Barnett, K. R. M. Leino, and W. Schulte. The Spec$\sharp$ programming system: An overview. In G. Barthe, L. Burdy, M. Huisman, J.-L. Lanet, and T. Muntean, editors, *Proc. CASSIS'04*, volume 3362 of *LNCS*, pages 49–69. Springer, 2005.
4. M. P. Bonacina and N. Dershowitz. Abstract canonical inference. *ACM TOCL*, 8(1):180–208, 2007.
5. M. P. Bonacina and M. Echenim. Rewrite-based satisfiability procedures for recursive data structures. In B. Cook and R. Sebastiani, editors, *Proc. 4th PDPAR Workshop, FLoC 2006*, volume 174(8) of *ENTCS*, pages 55–70. Elsevier, 2007.
6. M. P. Bonacina and M. Echenim. On variable-inactivity and polynomial T-satisfiability procedures. *J. Logic and Comput.*, 18(1):77–96, 2008.
7. M. P. Bonacina and M. Echenim. Theory decision by decomposition. *J. Symb. Comput.*, pages 1–42, To appear.
8. M. P. Bonacina, S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decidability and undecidability results for Nelson-Oppen and rewrite-based decision procedures. In U. Furbach and N. Shankar, editors, *Proc. 3rd IJCAR*, volume 4130 of *LNAI*, pages 513–527. Springer, 2006.
9. R. Caferra, A. Leitsch, and N. Peltier. *Automated Model Building.* Kluwer Academic Publishers, Amsterdam, 2004.
10. D. Cantone and C. G. Zarba. A decision procedure for monotone functions over bounded and complete lattices. In H. de Swart, editor, *Proc. TARSKI II*, volume 4342 of *LNAI*, pages 318–333. Springer, 2006.
11. L. de Moura and N. Bjørner. Engineering DPLL(T) + saturation. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proc. 4th IJCAR*, volume 5195 of *LNAI*, pages 475–490. Springer, 2008.
12. L. de Moura and N. Bjørner. Model-based theory combination. In S. Krstić and A. Oliveras, editors, *Proc. 5th SMT Workshop, CAV 2007*, volume 198(2) of *ENTCS*, pages 37–49. Elsevier, 2008.
13. C. Flanagan, K. R. M. Leino, M. Lillibridge, G. Nelson, J. B. Saxe, and R. Stata. Extended static checking for Java. In *Proc. PLDI*, pages 234–245, 2002.
14. J. Gallier, P. Narendran, D. A. Plaisted, S. Raatz, and W. Snyder. Finding canonical rewriting systems equivalent to a finite set of ground equations in polynomial time. *J. ACM*, 40(1):1–16, 1993.
15. J. Y. Halpern. Presburger Arithmetic with unary predicates is Π^1_1 Complete. *J. Symb. Logic*, 56:637–642, 1991.
16. S. Jacobs. Incremental instance generation in local reasoning. In *Notes 1st CEDAR Workshop, IJCAR 2008*, pages 47–62, 2008.
17. C. A. Lynch. Unsound theorem proving. In J. Marcinkowski and A. Tarlecki, editors, *Proc. CSL'04*, volume 3210 of *LNCS*, pages 473–487. Springer, 2004.
18. H. M. MacNeille. Partially ordered sets. In *Transactions of the American Mathematical Society*, volume 42, pages 416–460, 1937.
19. W. W. McCune. Otter 3.3 reference manual. Technical Report ANL/MCS-TM-263, MCS Division, Argonne National Laboratory, Argonne, IL, USA, 2003.
20. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM TOPLAS*, 1(2):245–257, 1979.
21. W. Snyder. A fast algorithm for generating reduced ground rewriting systems from a set of ground equations. *J. Symb. Comput.*, 1992.
22. V. Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In R. Nieuwenhuis, editor, *Proc. 20th CADE*, volume 3632 of *LNAI*, pages 219–234. Springer, 2005.