J Autom Reasoning (2011) 47:161–189
DOI 10.1007/s10817-010-9213-y

On Deciding Satisfiability by Theorem Proving with Speculative Inferences

Maria Paola Bonacina · Christopher A. Lynch · Leonardo de Moura

Received: 22 February 2010 / Accepted: 1 December 2010 / Published online: 22 December 2010

Abstract Applications in software verification often require determining the satisfiability of first-order formulae with respect to 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 to be highly scalable, efficient and suitable for integrated theory reasoning. Inference systems with resolution and superposition are strong at reasoning with equalities, universally quantified variables, and Horn clauses. We describe a theorem-proving method that tightly integrates superposition-based inference system and SMT solver. The combination is refutationally complete if background theory symbols only occur in ground formulae, and non-ground clauses are variable-inactive. Termination is enforced by introducing additional axioms as hypotheses. The system detects any unsoundness introduced by these speculative inferences and recovers from it.

Keywords Program checking · Theorem proving · Satisfiability modulo theories · Combination of theories

The first author was supported in part by grant no. 2007-9E5KM8 of the Ministero dell'Istruzione Università e Ricerca.

M. P. Bonacina (✉)
Dipartimento di Informatica, Università degli Studi di Verona,
Strada Le Grazie 15, 37134 Verona, Italy
e-mail: mariapaola.bonacina@univr.it

C. A. Lynch
Department of Mathematics and Computer Science, Clarkson University,
Potsdam, NY 13699-5815, USA
e-mail: clynch@clarkson.edu

L. de Moura
Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
e-mail: leonardo@microsoft.com

1 Introduction

Applications in software verification have benefited greatly from recent advances in automated reasoning. Reasoning about programs often requires determining the satisfiability of first-order formulae with respect to some background theories. In numerous contexts in software verification, quantifiers are also 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 formulae. Thus, many verification problems consist in determining the satisfiability of a set of formulae $\mathscr{R} \uplus P$ modulo a background theory $\mathscr{T}$, where $\mathscr{R}$ is a set of non-ground clauses without occurrences of $\mathscr{T}$-symbols, and P is a large ground formula, or set of ground clauses, that typically contains $\mathscr{T}$-symbols. The set of formulae $\mathscr{R}$ can be viewed as the axiomatization of an application specific theory. The background theory $\mathscr{T}$ is a combination $\mathscr{T} = \bigcup_{i=1}^{n} \mathscr{T}_i$ of theories $\mathscr{T}_i$, $1 \leq i \leq n$, commonly used in hardware and software verification, such as linear arithmetic.

Satisfiability Modulo Theories (SMT) solvers have proven highly scalable, efficient and suitable for integrated theory reasoning. Most SMT solvers are restricted to ground formulae, and integrate the Davis-Putnam-Logemann-Loveland procedure (DPLL) for propositional logic [25, 26], with satellite solvers for ground satisfiability problems in the theories $\mathscr{T}_i$, $1 \leq i \leq n$, that are therefore built into the SMT solver. The resulting integration is called DPLL($\mathscr{T}$), where $\mathscr{T} = \bigcup_{i=1}^{n} \mathscr{T}_i$. General treatments appeared in [55, 61]. For quantifiers, there are situations where the needed instances of universally quantified variables can be computed without loss of completeness: for instance, for certain fragments of the theories of arrays [17, 21] and pointers [52], or for local theory extensions [43, 63]. Otherwise, techniques to guess how to instantiate variables, based on heuristics and user annotations, were investigated [27, 34, 39]. They are known as "triggering," because the terms to be instantiated are called "triggers." These techniques can be very efficient when they succeed, but they require expensive user guidance, and their incompleteness causes false positives in the program verification tools that use the SMT-solver.

In comparison with SMT solvers, generic inference systems based on superposition and resolution are refutationally complete for first-order logic with equality; they are strong at reasoning with equalities, universally quantified variables, and Horn clauses. Available surveys include [10, 48, 56]. A standard superposition-based inference system was proved to terminate and hence to be a satisfiability procedure for several theories of data structures, including arrays and recursive data structures, and their combinations [3–5, 15].

The DPLL(Γ) system of [28] integrates an SMT solver with a generic inference system Γ based on superposition and resolution. The DPLL engine works by building a candidate model: if the construction succeeds, it returns satisfiable; if it fails definitely, it returns unsatisfiable. The inference system Γ works by deducing clauses from clauses and removing redundant clauses: if it generates the empty clause $\square$, that represents contradiction, it returns unsatisfiable. The key to their integration is that the literals in the candidate model built by the DPLL engine can occur as premises of Γ-inferences. The resulting system aims at uniting the strengths of SMT solvers—propositional efficiency, fast theory solvers, tight integration—with those of superposition-based theorem provers, especially general reasoning about quantifiers without recurring to incomplete heuristics.

However, in general, DPLL(Γ) is not refutationally complete, when both $\mathscr{T}$ and $\mathscr{R}$ are not empty, even when $\mathscr{T}$-symbols do not occur in $\mathscr{R}$. For example, assume $\mathscr{R} = \{x \simeq a \vee x \simeq b\}$ and $P = \emptyset$, and the background theory $\mathscr{T}$ is arithmetic. The clause $x \simeq a \vee x \simeq b$ implies that any model has a domain with at most two elements, which is clearly incompatible with any model for arithmetic, that requires an infinite domain. In other words, DPLL(Γ) does not have a way to detect unsatisfiability due to the lack of models with infinite domain. A first contribution of this article is a revised version of DPLL(Γ), named DPLL($\Gamma + \mathscr{T}$), with sufficient conditions to make it refutationally complete when $\mathscr{T}$ is not empty.

DPLL($\Gamma + \mathscr{T}$) has to combine the built-in theories in $\mathscr{T} = \bigcup_{i=1}^{n} \mathscr{T}_i$ and the axiomatized theory $\mathscr{R}$. Combination of built-in theories is usually done by the *equality sharing* method of [53], later dubbed "Nelson–Oppen scheme" from the names of the authors. This method has three requirements. First, the theories cannot share function symbols. Second, each $\mathscr{T}_i$ must be *stably infinite*. A theory $\mathscr{T}_i$ is stably infinite if every $\mathscr{T}_i$-satisfiable ground formula has a $\mathscr{T}_i$-model with domain of infinite cardinality. Third, every $\mathscr{T}_i$-solver must be capable of generating all entailed disjunctions of equalities between shared constants. The third requirement is relaxed in *model-based theory combination* [29], which is a version of equality sharing, where it is sufficient that each $\mathscr{T}_i$-solver generates the equalities between shared constants that are true in the current candidate $\mathscr{T}_i$-model. Thus, this method requires that each $\mathscr{T}_i$-solver generates a candidate model. On the other hand, combination of axiomatized theories in a superposition-based engine requires that they do not share function symbols and are *variable-inactive*: under these hypotheses, if the superposition-based system terminates on satisfiability problems in each theory, it also terminates on satisfiability problems in their union [3, 4]. A second contribution of this article is to explain how to apply known results on variable inactivity [4, 17, 18] to combine built-in theories by model-based theory combination and axiomatized theories in DPLL($\Gamma + \mathscr{T}$).

In software verification, during development time, several conjectures are false because of errors 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 formulae of linear arithmetic with uninterpreted functions [41]. 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. In that context, the symbol $\sqsubseteq$ represents a subtype relationship, and the monadic function f represents a type constructor, such as `Array-of`.

As an example, assume that the axiomatization contains a monotonicity axiom

$$\neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y).$$

Unrestricted resolution would resolve it with itself, that is, with a variant

$$\neg(x' \sqsubseteq y') \vee f(x') \sqsubseteq f(y')$$

to generate $\neg(x \sqsubseteq y) \vee f^2(x) \sqsubseteq f^2(y)$, hence the infinite series

$$\{\neg(x \sqsubseteq y) \vee f^i(x) \sqsubseteq f^i(y)\}_{i \geq 0},$$

where at each step the original axiom resolves with $\neg(x \sqsubseteq y) \vee f^{i-1}(x) \sqsubseteq f^{i-1}(y)$ to yield its successor. A selection function that selects negative literals prevents all these inferences, because $f(x') \sqsubseteq f(y')$ cannot resolve with $\neg(x \sqsubseteq y)$, if $\neg(x' \sqsubseteq y')$ is selected. However, even resolution with negative selection generates an infinite series

$$\{f^i(a) \sqsubseteq f^i(b)\}_{i \geq 0}$$

from monotonicity and each literal $a \sqsubseteq b$ in the 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 contribution of this article is a new version of DPLL($\Gamma + \mathscr{T}$) with *speculative inferences*, a feature suggested by [49]. The idea is to allow the prover, or the experimenter, to guess additional axioms, that avoid infinitary behaviors such as that induced by the monotonicity axiom. If the additional axioms are a cause of unsoundness, by turning a satisfiable set into an unsatisfiable one, the prover detects it and recovers from it automatically. The resulting method yields decision procedures for several axiomatizations of type systems that are relevant to software verification.

This article is organized as follows. Section 2 provides the background. Section 3 shows how to apply previous results on superposition and variable-inactive theories in DPLL($\Gamma + \mathscr{T}$); it states the requirements on the problem $\mathscr{R} \uplus P$ and the background theory $\mathscr{T}$, setting the stage for the sequel. Section 4 presents the new DPLL($\Gamma + \mathscr{T}$) system with speculative inferences. Section 6 introduces the notion of *essentially finite theories*—a generalization of the *finite model property*—exhibits essentially finite theories, and shows how DPLL($\Gamma + \mathscr{T}$) with speculative inferences yields a decision procedure for them and their combinations. Section 7 concludes with summary of the results, comparison with related work, and directions for further research. A short version of this article appeared in [20].

2 Background

We assume basic notions from logic used in theorem proving. Let Σ be a *signature* consisting of a set of *function* and *predicate* symbols, each with its *arity*, denoted by *arity*(f), for symbol f. We call 0-arity function symbols *constant* symbols, and use a, b, c, d for constants, f, g, h for non-constant function symbols, p for a predicate symbol, x, y, z, u for variables, $\simeq$ for equality, and $\bowtie$ for either $\simeq$ or $\not\simeq$: these three symbols are symmetric. Two signatures are *disjoint* if they share no function or predicate symbol other than $\simeq$. Terms, literals, clauses, sentences and formulæ are defined as usual. A clause (i.e., a disjunction of literals) is *positive* if all its literals are; it is *Horn* if it has at most one positive literal. We use t, s, l, r for terms, l, m for literals, C, D for clauses, $\Box$ for the empty clause, which denotes contradiction,

and F, N, P, S for sets of clauses. $Var(l)$ is the set of variables occurring in l. The notation $l[t]$ means that t appears as subterm of l. A first-order Σ-theory is *presented*, or axiomatized, by a set of Σ-sentences. Two theories are *disjoint* if their signatures are. We reserve calligraphic letters, such as $\mathscr{T}$ and $\mathscr{R}$, for presentations of theories. A *Horn presentation* is a set of non-negative Horn clauses. *Defined*, or *interpreted*, symbols are those symbols whose interpretation is restricted to the models of a theory, whereas *free*, or *uninterpreted*, symbols are those symbols whose interpretation is unrestricted.

A Σ-*structure* Φ consists of a non-empty universe, or domain, $|\Phi|$, and an interpretation for variables and symbols in Σ. We use v, w for elements of $|\Phi|$. For each $f \in \Sigma$, the interpretation of f is denoted by $\Phi(f)$. For a function symbol f with $arity(f) = n$, $\Phi(f)$ is an n-ary function on $|\Phi|$ with $\mathsf{range}(\Phi(f)) = \{w \mid \exists v \in |\Phi|,\ \Phi(f)(v) = w\}$. For a predicate symbol p with $arity(p) = n$, $\Phi(p)$ is a subset of $|\Phi|^n$. The interpretation of a term t is denoted by $\Phi(t)$. If t is a variable or constant, $\Phi(t)$ is an element in $|\Phi|$. Otherwise, $\Phi(f(t_1, \ldots, t_n)) = \Phi(f)(\Phi(t_1), \ldots, \Phi(t_n))$. If S is a set of terms, $\Phi(S) = \{\Phi(t) \mid t \in S\}$. Satisfaction $\Phi \models C$ is defined as usual; if $\Phi \models C$, then Φ is a *model* of C. For refutational completeness in first-order theorem proving it is sufficient to consider *Herbrand interpretations*, where the domain is the Herbrand universe and constant and function symbols are interpreted as themselves.

An *inference system* Γ is a set of inference rules. *Ordering-based* inference systems use an ordering $\succ$ on terms and literals to restrict *expansion inferences*, that expand the existing set by generating clauses, and to define *contraction inferences*, that contract the set by removing clauses. This ordering is assumed to be a *complete simplification ordering*: it is *stable* (if $s \succ t$ then $s\sigma \succ t\sigma$ for all substitutions σ), *monotone* (if $s \succ t$ then $l[s] \succ l[t]$ for all l), it has the *subterm property* ($l[t] \succ t$ for all t and $l \neq t$), hence it is *well-founded* (there is no infinite decreasing chain $t_1 \succ t_2 \succ \ldots t_i \succ t_{i+1} \succ \ldots$) [31], and it is *total* on ground terms and literals (if $s \neq t$, then either $s \succ t$ or $t \succ s$). The ordering is extended to equations and clauses by multiset extension, which preserves well-foundedness [33]. An *inference rule* with n premises has a *main premise* and $n - 1$ *side* premises. For an expansion rule, the main premise yields the conclusion in the context of the side premises; such a rule is *sound* if the conclusion is a logical consequence of the premises. For a contraction rule, the main premise is reduced to the conclusion or removed; such a rule is *sound* if the main premise is a logical consequence of side premises and conclusion, if present. Premises and conclusion of an inference γ are denoted by $\mathsf{P}(\gamma)$ and $\mathsf{C}(\gamma)$, respectively. We write $\gamma \in \Gamma$ to say that γ is an application of an inference rule in Γ.

Clauses deleted by contraction are *redundant*. Redundancy is defined based on well-founded orderings on clauses, whereby a ground clause is redundant in a set of clauses, if it is entailed by smaller ground clauses in the set, and a clause is redundant if all its ground instances are [6], or well-founded orderings on proofs, whereby a clause is redundant in a set of clauses, if it does not affect its minimal proofs [19]. An inference is redundant if it uses or generates a redundant clause. A clause or inference that is not redundant is *irredundant*. A set of clauses N is *saturated with respect to* Γ, or Γ-*saturated*, if all Γ-inferences in N are redundant. Given an input set of clauses S_0, a Γ-*derivation* is a sequence $S_0 \vdash_\Gamma S_1 \vdash_\Gamma \ldots S_i \vdash_\Gamma S_{i+1} \vdash_\Gamma \ldots$ where at each step S_{i+1} is derived from S_i by a Γ-inference; its *limit* is the set of *persistent clauses* $S_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} S_j$. A Γ-derivation is *fair*, if all expansion inferences from persistent irredundant premises are done eventually. If a Γ-derivation is fair, its limit

is saturated. Since Γ is non-deterministic, there may be more than one Γ-derivation from a given S_0. The combination of Γ with a *search plan*, that controls the choice of inferences, yields a deterministic *theorem-proving strategy*, or *proof procedure*, termed Γ-*strategy*, or Γ-*procedure*, whose derivation is unique given S_0. A strategy or procedure is fair if all its derivations are. A recent abstract treatment of these notions, with references to their history, appeared in [12].

Let I be a mapping, called a *model functor*, that assigns to each set of ground clauses N, not containing $\Box$, a Herbrand interpretation I_N, called the *candidate model*. A clause C is a counterexample for I_N if $I_N \not\models C$; C is a minimal counterexample if, in addition, there is no other counterexample D for I_N such that $C \succ D$. An inference system Γ has the *reduction property for counterexamples*, if for all sets N of clauses and 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. This property is used in proofs of refutational completeness since at least [6] according to the following standard:

Theorem 1 *If N is a Γ-saturated set of ground clauses and Γ has the reduction property for counterexamples, then N is unsatisfiable if and only if it contains $\Box$.*

Proof The "if" direction is trivial. To prove the "only if" direction, one proves its contrapositive: if N does not contain $\Box$, then it is satisfiable. By way of contradiction, assume that it is not. Then, for every candidate model I_N there is a counterexample in N. Let C be a minimal counterexample for I_N in N. By the reduction property for counterexamples, there is a smaller counterexample D, conclusion of a Γ-inference. If $D \in N$, then C would not have been minimal to begin with. If $D \notin N$, then N is not saturated, which contradicts the hypothesis. □

We do not assume a specific inference system: Γ is a parameter for DPLL($\Gamma + \mathscr{T}$). Since examples and proofs for specific theories mention concrete inference rules, and in order to make this article self-contained, typical expansion and contraction rules are collected in Fig. 1: in contraction, what is above the double inference line is replaced by what is below, whereas in expansion, as usual, what is below the single inference line is added to what is above. In resolution, l and l' are the literals *resolved upon*. In paramodulation, $l[s']$ $(C \vee l[s'])$ is the literal (clause) *paramodulated into*, and $s \simeq t$ $(D \vee s \simeq t)$ is the literal (clause) *paramodulated from*. The same terminology applies to superposition.[1] Resolution, paramodulation and superposition originally appeared in [59], [57] and [45], respectively. Since then, they were the object of decades of research: contemporary versions of these rules appeared in [6, 19, 42, 60]; more references and history can be found in [10–12, 48, 56].

In addition to the ordering, expansion inference rules can be restricted by a *selection function*, that selects negative literals [6]. A clause can have all, some, or none of its negative literals selected, depending on the selection function. In *resolution with negative selection*, *paramodulation with negative selection* and *superposition with negative selection*, the ordering constraint on the negative literal

[1] We use superposition when the literal paramodulated into is equational, and paramodulation otherwise. Other articles reserve superposition to unit clauses or positive equations.

resolved upon, on the literal paramodulated into (i.e., condition (iii) in Fig. 1) and on the literal superposed into (i.e., condition (v) in Fig. 1) is lifted, and replaced by two requirements: the negative literal resolved upon, the literal paramodulated into and the literal superposed into must be selected, and the other premise contains no selected literal. If some negative literal is selected for each clause containing one, one premise in each resolution, paramodulation or superposition inference will be a positive clause, yielding a *positive strategy*; if, in addition, all clauses are Horn, only positive unit clauses can be resolved upon or superposed or paramodulated from, yielding a *unit strategy* with *unit resolution* and *unit superposition* [32]. Contemporary

Resolution	$\frac{C \vee \neg l \quad D \vee l'}{(C \vee D)\sigma}$	$\forall m \in C : \neg l\sigma \not\preceq m\sigma,\ \forall m \in D : l'\sigma \not\preceq m\sigma$
Factoring	$\frac{C \vee l \vee l'}{(C \vee l)\sigma}$	$\forall m \in C : l\sigma \not\preceq m\sigma$
Paramodulation	$\frac{C \vee l[s'] \quad D \vee s \simeq t}{(C \vee D \vee l[t])\sigma}$	$(i),\ (ii),\ (iii)$
Superposition	$\frac{C \vee l[s'] \bowtie r \quad D \vee s \simeq t}{(C \vee D \vee l[t] \bowtie r)\sigma}$	$(i),\ (ii),\ (iv),\ (v)$
Reflection	$\frac{C \vee s' \not\simeq s}{C\sigma}$	$\forall l \in C : (s' \not\simeq s)\sigma \not\prec l\sigma$
Equational Factoring	$\frac{C \vee s \simeq t \vee s' \simeq t'}{(C \vee t \not\simeq t' \vee s \simeq t')\sigma}$	$(i),\ \forall l \in \{s' \simeq t'\} \cup C : (s \simeq t)\sigma \not\prec l\sigma$

where σ is the most general unifier (mgu) of l and l' in resolution and factoring, and of s and s' in the other rules; s' is not a variable in paramodulation and superposition, and the following abbreviations hold:

(i) is $s\sigma \not\preceq t\sigma$,
(ii) is $\forall m \in D : (s \simeq t)\sigma \not\preceq m\sigma$,
(iii) is $\forall m \in C : l[s']\sigma \not\preceq m\sigma$,
(iv) is $l[s']\sigma \not\preceq r\sigma$, and
(v) is $\forall m \in C : (l[s'] \bowtie r)\sigma \not\preceq m\sigma$.

Strict Subsumption	$\frac{C \quad D}{\quad C \quad}$	$D \rhd C$
Simplification	$\frac{C[l] \quad s \simeq t}{C[t\sigma], \quad s \simeq t}$	$l = s\sigma, \quad s\sigma \succ t\sigma, \quad C[l] \succ (s \simeq t)\sigma$
Deletion	$\frac{C \vee t \simeq t}{\quad}$	

where $D \rhd C$ if $D \trianglerighteq C$ and $C \not\trianglerighteq D$; and $D \trianglerighteq C$ if $C\sigma \subseteq D$ (as multisets) for some substitution σ. In practice, theorem provers also apply subsumption of variants (if $D \trianglerighteq C$ and $C \trianglerighteq D$, the oldest clause is retained) and tautology deletion (that removes clauses such as $C \vee l \vee \neg l$).

Fig. 1 Sample expansion and contraction rules: in expansion what is below the inference line is added to the clause set that contains what is above the inference line; in contraction what is above the double inference line is removed and what is below is added

theorem provers, including the implementation of DPLL($\Gamma + \mathscr{T}$) on top of Z3 [30], use resolution with negative selection to implement *hyperresolution* [58]. In hyperresolution, the side premises, termed *satellites*, are positive clauses that resolve away all negative literals in the main premise, termed *nucleus*, generating a positive clause, in a single step with a simultaneous unifier of all pairs of literals resolved upon.[2] A selection function that selects some negative literal in each clause containing one induces resolution to simulate hyperresolution as a macro inference involving several steps of resolution. In this article, hyperresolution is realized via resolution with negative selection.

3 Variable Inactivity in DPLL($\Gamma + \mathscr{T}$)

In this section we see how previous results from [4, 17, 18] can be imported into DPLL($\Gamma + \mathscr{T}$) to combine a built-in theory $\mathscr{T}$ and an axiomatized theory $\mathscr{R}$, where both $\mathscr{T}$ and $\mathscr{R}$ can be themselves unions of theories. In a purely rewrite-based approach there is no $\mathscr{T}$ and all axioms are part of the input in $\mathscr{R}$. The ordering $\succ$ of Γ is required to be *good* [4, 16], meaning that $t \succ c$ for all ground compound term t and constant c. A fair Γ-strategy is shown to be an $\mathscr{R}$-satisfiability procedure, by showing that it is guaranteed to terminate on $\mathscr{R}$-satisfiability problems $\mathscr{R} \uplus S$, where S is a set of ground unit $\mathscr{R}$-clauses. Variable-inactivity was introduced in [3, 4]:

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.

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

It was proved in [4] (cf. Theorem 4.1 and Corollary 3) that termination is *modular*:

Theorem 2 [4] *Let $\mathscr{R}_1, \ldots, \mathscr{R}_n$ be disjoint and variable-inactive for Γ, and let $\mathscr{R} = \bigcup_{i=1}^{n} \mathscr{R}_i$. If a fair Γ-strategy terminates on $\mathscr{R}_i$-satisfiability problems, for $1 \leq i \leq n$, then it terminates also on $\mathscr{R}$-satisfiability problems.*

A *cardinality constraint* 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. The following key lemma was proved in [18] (cf. Lemma 5.2):

Lemma 1 [18] *If S_0 is a finite satisfiable set of clauses, then S_0 admits no infinite models if and only if the limit S_∞ of a fair Γ-derivation from S_0 contains a cardinality constraint.*[3]

[2] This instance of the rule is called *positive hyperresolution*. The dual rule named *negative hyperresolution* operates in the same way with polarities exchanged. Since selection functions are defined to select negative literals, negative hyperresolution falls outside of this discussion.

[3] Lemma 5.2 in [18] 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.

It follows that (cf. Theorem 4.5 in [4]):

Theorem 3 [4] *If $\mathscr{R}$ is variable-inactive for* Γ*, then it is stably-infinite.*

Thus, Γ reveals the lack of stable infiniteness by generating a cardinality constraint. The original versions of Theorem 2, Lemma 1 and Theorem 3 were proved in a context where equality was the only predicate and superposition the main expansion inference rule of Γ. It is trivial to extend them to the case where the signature of $\mathscr{R}$ introduces predicate symbols other than equality, and Γ features also resolution and paramodulation. For instance, for Theorem 2, the essence of the proof is to show that there are only finitely many inferences across theories: disjointness of the signatures prevents not only superpositions, but also paramodulations from compound terms, and resolutions; variable inactivity prevents not only superpositions, but also paramodulations from variables; thus, the only inferences across theories are superpositions and paramodulations from shared constants, that are finitely many.

It is useful to import results from the rewrite-based approach to DPLL($\Gamma + \mathscr{T}$), applied to a problem $\mathscr{R} \uplus P$ modulo $\mathscr{T}$, because DPLL($\Gamma + \mathscr{T}$) uses Γ as an $\mathscr{R}$-solver applied to $\mathscr{R}$-satisfiability problems $\mathscr{R} \uplus S$, where S is a set of ground unit $\mathscr{R}$-clauses. The initial set of ground clauses P typically contains also $\mathscr{T}$-symbols. However, P is subject to *purification*, which is a standard step in the Nelson-Oppen method. This transformation, also known as *separation* [37], separates occurrences of function symbols from different signatures occurring in ground terms, by introducing new constant symbols. For example, $f(g(a)) \simeq b$, where f and g belong to different signatures, becomes $f(c) \simeq b \wedge g(a) \simeq c$, where c is new. Since only constants are introduced, the set remains ground. Thus, P is transformed in two disjoint sets P_1 and P_2, where P_1 contains only $\mathscr{R}$-symbols and P_2 only $\mathscr{T}$-symbols. Since a key feature of DPLL($\Gamma + \mathscr{T}$) is that Γ deals only with *non-ground clauses* and *ground unit clauses*, it is indeed the case that Γ works on an $\mathscr{R}$-satisfiability problems $\mathscr{R} \uplus S$: initially, S will be the subset of unit clauses from P_1.

DPLL($\Gamma + \mathscr{T}$) needs to combine $\mathscr{T}_1, \ldots, \mathscr{T}_n, \mathscr{R}$ in the Nelson-Oppen scheme, which requires that the theories are disjoint, stably infinite, and each solver generates all entailed disjunctions of equalities between shared constants. We assume that $\mathscr{T}_1, \ldots, \mathscr{T}_n$ satisfy these requirements and that $\mathscr{R}$ is disjoint from each of them. For stable infiniteness of $\mathscr{R}$, we require that $\mathscr{R}$ is variable-inactive and apply Theorem 3. In practice, this condition is checked dynamically: in the implementation of DPLL($\Gamma + \mathscr{T}$) on top of Z3, the superposition-based engine is equipped with a test that detects the generation of variable-inactive clauses, hence cardinality constraints, and discovers whether $\mathscr{R}$ is not stably infinite. Such a test also excludes upfront a situation such as $\mathscr{R} = \{x \simeq a \vee x \simeq b\}$ of the example in Section 1. For the generation of disjunctions of equalities between shared constants by the $\mathscr{R}$-solver Γ, the fairness of the Γ-derivation ensures that every theorem is implied by some generated formulae.[4] An explicit proof that the superposition-based engine generates formulae that entail all disjunctions of equalities between constants in the axiomatized theory was given in [17] (cf. Theorem 71). If contraction is also done systematically, only irredundant clauses generated by Γ are kept and passed to the DPLL($\mathscr{T}$) core.

[4] Fairness guarantees even more: every theorem has a minimal proof in the limit; see [12] for details.

The following definition summarizes the problem requirements for the sequel:

Definition 3 A set of formulae $S = \mathscr{R} \uplus P$ is *smooth* with respect to a background theory $\mathscr{T} = \bigcup_{i=1}^{n} \mathscr{T}_i$, or $\mathscr{T}$*-smooth* for short, if

- $\mathscr{T}_1, \ldots, \mathscr{T}_n$ and $\mathscr{R}$ are pairwise disjoint,
- $\mathscr{T}_1, \ldots, \mathscr{T}_n$ are stably infinite,
- $\mathscr{R}$ is variable-inactive, and
- P is a set of ground formulae $P_1 \uplus P_2$, where P_1 contains only $\mathscr{R}$-symbols, and P_2 only $\mathscr{T}$-symbols.

Note that uninterpreted symbols are $\mathscr{R}$-symbols. In summary, variable inactivity is an ingredient for: (1) modularity of termination of Γ, when $\mathscr{R}$ is a union of axiomatized theories [4]; (2) stable infiniteness of $\mathscr{R}$ [18], hence combination of axiomatized theories and built-in theories; (3) refutational completeness of DPLL($\Gamma + \mathscr{T}$) when both $\mathscr{T}$ and $\mathscr{R}$ are not empty (cf. Theorem 4 in the next section).

4 A New DPLL($\Gamma + \mathscr{T}$) System with Speculative Inferences

DPLL($\Gamma + \mathscr{T}$) works on *hypothetical clauses* of the form $H \triangleright C$, where C is a clause, and the *hypothesis* H is a set of ground literals. The hypothesis is interpreted as a conjunction, and a hypothetical clause $(l_1 \wedge \ldots \wedge l_n) \triangleright (l'_1 \vee \ldots \vee l'_m)$ is interpreted as $\neg l_1 \vee \ldots \vee \neg l_n \vee l'_1 \vee \ldots \vee l'_m$. As we shall see, the literals in H come from the candidate model built by DPLL($\Gamma + \mathscr{T}$), and are the literals that C depends on, in the sense that they were used as premises to infer C by Γ-inferences. As it was done in [28] for DPLL(Γ), DPLL($\Gamma + \mathscr{T}$) is described as a *transition system* with two modes: *search mode* and *conflict resolution mode*.

In search mode, the state of the system has 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, possibly with a justification, 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 has no justification. An implied literal l_C is a literal l justified by a clause C: all other literals of C are false in M so that l needs to be true. 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*.

In conflict resolution mode, the state has the form $M \parallel F \parallel C$, where C is a ground clause whose literals are all false under M. Such a clause is in *conflict*. If C is $l_1 \vee \ldots \vee l_n$, then $\neg C$ is the formula $\neg l_1 \wedge \ldots \wedge \neg l_n$. We could state that C is in conflict by writing $M \models \neg C$. In DPLL($\Gamma + \mathscr{T}$), the DPLL engine accepts only propositional clauses, whereas the theory solvers accept ground first-order clauses and Γ accepts first-order clauses. To bridge this gap, an *abstraction function* maps first-order ground atoms to propositional atoms. Thus, it is customary to write $M \models_P \neg C$, read M "propositionally satisfies" $\neg C$, to say that M satisfies the propositional abstraction of $\neg C$.

Definition 4 Given an input set of clauses $S = \mathscr{R} \uplus P$, a *DPLL*$(\Gamma + \mathscr{T})$*-derivation* is a sequence of state transitions

$$\Delta_0 \Longrightarrow \Delta_1 \Longrightarrow \ldots \Delta_i \Longrightarrow \Delta_{i+1} \Longrightarrow \ldots$$

where $\forall i \geq 0$, Δ_i is of the form $M_i \parallel F_i$ or $M_i \parallel F_i \parallel C_i$, each transition is determined by a DPLL$(\Gamma + \mathscr{T})$-rule, and $\Delta_0 = \parallel F_0$ for $F_0 = \{\emptyset \rhd C \mid C \in S\}$.

In the sequel, we use C for $\emptyset \rhd C$, *clauses*(F) to denote the set $\{C \mid H \rhd C \in F\}$, *ngclauses*$(F)$ for the subset of non-ground clauses of *clauses*(F), *lits*(M) to denote the set of assigned literals, $lits_{\mathscr{R}}(M)$ for the subset of assigned $\mathscr{R}$-literals and *clauses*$^\star(M \parallel F)$ for *ngclauses*$(F) \cup lits_{\mathscr{R}}(M)$.

4.1 Speculative Inferences in DPLL$(\Gamma + \mathscr{T})$

In theorem proving applied to mathematics, most conjectures are true. Thus, it is customary to sacrifice completeness for efficiency, and retain soundness, which is necessary to attribute unsatisfiability to the input set of clauses if a proof is found. A traditional example is *deletion by weight* [51], where clauses that are too "heavy" are deleted. In theorem proving applied to verification, most conjectures are false. Thus, it was suggested in [49] 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 call *speculative* an inference that may turn out to be unsound.

We consider a single speculative inference rule: adding an arbitrary clause C. Such a step may be unsound because C may not be implied by the given set. This rule is simple, but can simulate different kinds of speculative 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$(\Gamma + \mathscr{T})$ with a *reversible* transition rule SpeculativeIntro for speculative inferences. Rather than merely adding a clause C, SpeculativeIntro introduces a hypothetical clause $\lceil C \rceil \rhd C$ into F and it adds $\lceil C \rceil$ to M: $\lceil C \rceil$ is a new propositional variable used as a label for clause C. By adding $\lceil C \rceil$ to M, the system records the fact that it is *guessing* C. SpeculativeIntro is *reversible*, because the system uses $\lceil C \rceil$ to track the consequences of having added C. The hypothetical clause $\lceil C \rceil \rhd C$ 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:

SpeculativeIntro

$$M \parallel F \quad \Longrightarrow \quad M\ \lceil C \rceil \parallel F,\ \lceil C \rceil \rhd C \qquad \textbf{if} \begin{cases} C \notin \mathit{clauses}(F), \\ \lceil C \rceil \ \mathit{is\ new}, \\ \lceil C \rceil, \neg\lceil C \rceil \notin M. \end{cases}$$

Note that $\lceil C \rceil$ is added to M as a decided literal. The first condition says that we do not guess a clause that we already have. The second condition requires $\lceil C \rceil$ to be a new symbol with respect to the initial signature. The third condition prevents the system from adding C, if it was already done ($\lceil C \rceil \in M$), or if the addition was already discovered to be inconsistent with the current partial model M ($\neg\lceil C \rceil \in M$).

4.2 Model-Based Theory Combination in DPLL($\Gamma + \mathscr{T}$)

In order to combine the theories in $\mathscr{T} = \bigcup_{i=1}^{n} \mathscr{T}_i$ and $\mathscr{R}$ in the Nelson-Oppen scheme, every $\mathscr{T}_i$-solver, $1 \leq i \leq n$, needs to communicate to the other theories, including $\mathscr{R}$, the (disjunctions of) equalities between shared constants entailed by $\mathscr{T}_i$ and P. The next transition rule takes care of this requirement, according to model-based theory combination [29]. We assume that every $\mathscr{T}_i$-solver builds a specific candidate $\mathscr{T}_i$-model for M, that we denote by $\mathsf{model}_i(M)$. For instance, solvers for linear arithmetic satisfy this requirement [35]. The idea is to inspect $\mathsf{model}_i(M)$ and propagate all the equalities it implies, hedging that they are consistent with the other theories, including $\mathscr{R}$. Since these equalities are *guesses*, if one of them turns out to be inconsistent, backtracking will be used to fix $\mathsf{model}_i(M)$. The rationale for this approach is practical: it is generally far less expensive to enumerate the equalities satisfied in a particular $\mathscr{T}_i$-model than those satisfied by all $\mathscr{T}_i$-models consistent with M; in most experiments, the number of equalities that are really relevant turns out to be small.

PropagateEq

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

for every theory $\mathscr{T}_i$, $1 \leq i \leq n$. Since the $\mathscr{T}_i$-solvers only deal with ground clauses, this rule treats only ground equalities, and therefore only ground terms that appear in F. The reason why it adds equalities between ground terms and not only between shared constants will be explained in relation to the Deduce rule.

4.3 The Core Transition Rules of DPLL($\Gamma + \mathscr{T}$)

Figure 2 reports the basic and theory propagation rules of DPLL($\Gamma + \mathscr{T}$) from [28].

The Decide rule is not concerned with literals in hypotheses, since such literals already come from M. The Deduce rule realizes the interface with Γ: assume γ is an inference 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 $H_1 \cup \ldots \cup H_m \cup \{l_{m+1}, \ldots, l_n\}$; if γ with premises $\mathsf{P}(\gamma) = \{C_1, \ldots, C_m, l_{m+1}, \ldots, l_n\}$ yields $\mathsf{C}(\gamma)$, the latter 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 differs from its predecessor in [28], named Deduce$^\sharp$, in the range of allowed premises from F. Deduce$^\sharp$ allowed Γ to use as premises $\mathit{lits}(M)$, and non-ground clauses and ground unit clauses from $\mathit{clauses}(F)$. Our Deduce allows Γ to use only $\mathit{clauses}^\star(M \parallel F) = \mathit{ngclauses}(F) \cup \mathit{lits}_{\mathscr{R}}(M)$. 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 $\mathit{lits}_{\mathscr{R}}(M)$. This is the reason why we let PropagateEq add equalities between ground terms and not only between shared constants.

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 conflict clause $H \triangleright C$ into a regular clause by negating its hypotheses, and puts the DPLL($\Gamma + \mathscr{T}$) 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 Backjump rule drives the DPLL($\Gamma + \mathscr{T}$) system back from conflict resolution mode to search mode, and it unassigns at least one decided literal, named l' in the rule definition in Fig. 2. A typical choice is that l' be the least recently decided literal that satisfies the conditions of the rule. 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.

Figure 3 reproduces from [28] the *contraction transitions* that import the contraction rules of Γ in DPLL($\Gamma + \mathscr{T}$): note that they apply only in search mode. These transitions and their explanation, that follows, refer to generic contraction rules

Decide

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

UnitPropagate

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

Deduce

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

Conflict

$$M \parallel F, H \triangleright C \implies M \parallel F, H \triangleright C \parallel \neg H \vee C \quad \textbf{if} \quad M \models_P \neg C$$

Explain

$$M \parallel F \parallel C \vee \bar{l} \implies M \parallel F \parallel \neg H \vee D \vee C \quad \textbf{if} \quad l_{H \triangleright (D \vee l)} \in M$$

Learn

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

Backjump

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

Unsat

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

T-Propagate

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

T-Conflict

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

Fig. 2 Basic and theory propagation rules of DPLL($\Gamma + \mathscr{T}$)

Delete

$$M \parallel F, H \triangleright C \quad \Longrightarrow \quad M \parallel F \qquad \textbf{if} \quad \begin{cases} \gamma_d(C, C_2, \ldots, l_n),\ n \geq 2 \\ level(H) \geq level(H') \end{cases}$$

Disable

$$M \parallel F, H \triangleright C \quad \Longrightarrow \quad M \parallel F, [H \triangleright C]_{level(H')} \qquad \textbf{if} \quad \begin{cases} \gamma_d(C, C_2, \ldots, l_n),\ n \geq 2 \\ level(H) < level(H') \end{cases}$$

Simplify

$$M \parallel F, H \triangleright C \quad \Longrightarrow \quad M \parallel F, (H \cup H') \triangleright C' \qquad \textbf{if} \quad \begin{cases} \gamma_s(C, C_2, \ldots, l_n, C'),\ n \geq 2 \\ level(H) \geq level(H') \end{cases}$$

Simplify-disable

$$M \parallel F, H \triangleright C \quad \Longrightarrow \quad M \parallel F, [H \triangleright C]_{level(H')}, (H \cup H') \triangleright C' \qquad \textbf{if} \quad \begin{cases} \gamma_s(C, C_2, \ldots, l_n, C'),\ n \geq 2 \\ level(H) < level(H') \end{cases}$$

Fig. 3 Contraction transitions of DPLL($\Gamma + \mathscr{T}$): the notation in the conditions is explained in the text

schemas, and not to the concrete contraction rules of Fig. 1, in order to show that this way of integrating contraction is general, and applies to the contraction rules of Fig. 1 as well as to others. Any sound contraction inference taking a single premise (e.g., tautology deletion) can be easily incorporated into DPLL($\Gamma + \mathscr{T}$). Given a hypothetical clause $H \triangleright C$, such a rule is just applied to C. Contraction rules with more than one premise need special treatment. We use $\gamma_d(C, C_2, \ldots, C_m)$ to denote the application of a generic sound deletion rule

$$\frac{C, C_2, \ldots, C_m}{C_2, \ldots C_m}$$

where a redundant main premise C is deleted, and $\gamma_s(C, C_2, \ldots, C_m, C')$ to denote the application of a generic sound simplification rule

$$\frac{C, C_2, \ldots, C_m}{C', C_2, \ldots C_m}$$

where a redundant main premise C is replaced by C'. DPLL($\Gamma + \mathscr{T}$) assigns a *scope level* to each literal in M:

Definition 5 The *scope level* of a literal l, denoted $level(l)$, in $M\ l\ M'$, is equal to the number of decided literals in $M\ l$. The *scope level* of a set of literals H is

$$level(H) = \begin{cases} max\{level(l) \mid l \in H\} & \text{if } H \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

A contraction inference γ from Γ is generalized to hypothetical clauses as follows: given main premise $H \triangleright C$, taken from $ngclauses(F)$, and side premises $H_2 \triangleright C_2, \ldots, H_m \triangleright C_m$, $l_{m+1}, \ldots, l_n$, taken from $ngclauses(F)$ and $lits_{\mathscr{R}}(M)$, respectively, let $H' = H_2 \cup \ldots \cup H_m \cup \{l_{m+1}, \ldots, l_n\}$. Assume that γ has premises $C, C_2, \ldots, C_m, l_{m+1}, \ldots, l_n$. First, for a simplification γ_s, $H \triangleright C$ is replaced by

$(H \cup H') \triangleright C'$. Second, for both γ_d and γ_s, $H \triangleright C$ is deleted only if $level(H) \geq level(H')$. Indeed, this condition prevents the situation where backjumping removes side premises (e.g., simplifiers or subsumers) before removing the main premise $H \triangleright C$ (i.e., the simplified or subsumed clause). Such a situation must be prevented, because otherwise the system would reach an unsound state, where $H \triangleright C$ was deleted, but the clauses that made it redundant and justified its deletion are no longer there. For this reason, if $level(H) < level(H')$, then $H \triangleright C$ is only *disabled*. In Fig. 3 a disabled clause is surrounded by square brackets and bears as subscript the level of the set of side premises that disabled it. A disabled clause is not deleted, but it is not used as premise. When $level(H')$ is backjumped, all disabled clauses with subscript $level(H')$ will be re-enabled and will be available again as premises.

4.4 Refutational Completeness of DPLL($\Gamma + \mathscr{T}$)

It was proved in [28] that DPLL(Γ) is refutationally complete when $\mathscr{T}$ is empty. We prove a stronger result for the case where both $\mathscr{T}$ and $\mathscr{R}$ are not empty. We start with definitions that adapt to DPLL($\Gamma + \mathscr{T}$) the classical notions of redundancy, fairness and saturation (cf. Section 2). We use Γ-*based transitions* for Deduce and the contraction transitions of Fig. 3.

Definition 6 A Γ-based transition is *redundant* in state $M \parallel F$ if the corresponding Γ-inference is redundant in $clauses^\star(M \parallel F)$.

Note that Γ-based transitions apply only in search mode.

Definition 7 A DPLL($\Gamma + \mathscr{T}$)-derivation is *fair* if all applicable transitions are applied eventually, except SpeculativeIntro and redundant Γ-based transitions.

We recall that: (1) contraction rules are part of Γ; (2) Γ, and therefore contraction rules, only sees $clauses^\star(M \parallel F) = ngclauses(F) \cup lits_{\mathscr{R}}(M)$; (3) contraction inferences delete only clauses in $ngclauses(F)$. All other transitions do not use $ngclauses(F)$ and are therefore sheltered from contraction. Thus, the only transitions that are affected by contraction, and for which we need to stipulate that only irredundant inferences are considered, are Γ-based transitions.

Definition 8 A DPLL($\Gamma + \mathscr{T}$) state is *saturated* if it is

- either *unsat*
- or a state $M \parallel F$ such that the only applicable transitions are SpeculativeIntro transitions or redundant Γ-based transitions.

Clearly, a fair derivation yields a saturated state eventually. In order to prove refutational completeness – whenever the input set S is unsatisfiable, DPLL($\Gamma + \mathscr{T}$) reaches the *unsat* state – we prove as usual its contrapositive:

Theorem 4 *If the initial set of clauses $S = \mathscr{R} \uplus P$ is $\mathscr{T}$-smooth, and Γ has the reduction property for counterexamples, whenever DPLL($\Gamma + \mathscr{T}$) reaches a saturated state $M \parallel F$, the input set S is satisfiable modulo $\mathscr{T}$.*

Proof We need to show that if $M \parallel F$ is saturated, then $clauses(F) \cup lits(M)$ is satisfiable. Satisfiability of S will follow, because the transition rules in DPLL($\Gamma + \mathscr{T}$) are sound and therefore preserve satisfiability. Let N be $clauses(F) \cup lits(M)$. The set N has the form $\mathscr{R}' \uplus M_1 \uplus G_1 \uplus M_2 \uplus G_2$, where $\mathscr{R}'$ contains non-ground clauses (i.e., $\mathscr{R}' = ngclauses(F)$), $M_1 \uplus G_1 \uplus M_2 \uplus G_2$ is ground, $M_1 \uplus M_2 = lits(M)$, $\mathscr{R}' \uplus G_1 \uplus G_2 = clauses(F)$, $G_1 \uplus M_1$ contains only $\mathscr{R}$-symbols (i.e., $M_1 = lits_{\mathscr{R}}(M)$), and $G_2 \uplus M_2$ contains only $\mathscr{T}$-symbols.

- We consider first $\mathscr{R}' \uplus G_1 \uplus M_1$. In a standard proof of completeness for Γ alone, we would have that $\mathscr{R}' \uplus G_1 \uplus M_1$ is Γ-saturated, because the Γ-derivation is fair. For DPLL($\Gamma + \mathscr{T}$) we need to show that $\mathscr{R}' \uplus G_1 \uplus M_1$ is Γ-saturated, even if Γ-based transitions do not use G_1. Since $M \parallel F$ is saturated, for every clause $C \in G_i$, for $i \in \{1, 2\}$, there is a literal l of C in M_i. Indeed, if this were not the case, the Decide rule could apply, violating the hypothesis that $M \parallel F$ is saturated. Thus, every clause $C \in G_1$ is subsumed by a literal in M_1, therefore it is redundant in $M_1 \uplus G_1$, and every Γ-based transition that uses C is redundant. Then, $\mathscr{R}' \uplus M_1$ alone is Γ-saturated: if it were not, an irredundant Γ-based transition could apply, violating the hypothesis that $M \parallel F$ is saturated. It follows that $\mathscr{R}' \uplus G_1 \uplus M_1$ is Γ-saturated. Since $\mathscr{R}' \uplus G_1 \uplus M_1$ does not contain $\Box$, and Γ has the reduction property for counterexamples, $\mathscr{R}' \uplus G_1 \uplus M_1$ is satisfiable by Theorem 1.
- We consider next $G_2 \uplus M_2$: this set is satisfiable modulo $\mathscr{T}$, because if it were not, the T-Conflict rule would apply, and $M \parallel F$ would not be saturated.
- By the hypothesis that the initial set $S = \mathscr{R} \uplus P$ is $\mathscr{T}$-smooth (cf. Definition 3), $\mathscr{R}$ is variable-inactive. By Definition 2, $\mathscr{R}'$, which is derived from $\mathscr{R}$ and ground unit $\mathscr{R}$-clauses, is also variable-inactive, hence stably infinite by Theorem 3. Thus, $\mathscr{R}' \uplus G_1 \uplus M_1$ has a model with infinite domain. Again by the hypothesis that $\mathscr{R} \uplus P$ is $\mathscr{T}$-smooth, $\mathscr{T}$ is a union of stably infinite theories. Thus, $G_2 \uplus M_2$ has a $\mathscr{T}$-model with infinite domain. Since all the requirements for a Nelson-Oppen combination are fulfilled, these two models can be combined in a $\mathscr{T}$-model of N by the completeness of equality sharing, establishing that $clauses(F) \cup lits(M)$ is satisfiable. □

All inference systems considered in the sequel have the reduction property for counterexamples [6, 56]. The proof of Theorem 4 shows that the integration of the components in DPLL($\Gamma + \mathscr{T}$) is designed in such a modular way that its completeness descends from the completeness of its components.

5 Towards Decision Procedures: DPLL($\Gamma + \mathscr{T}$)-Strategies

The combination of the transition system DPLL($\Gamma + \mathscr{T}$) with a *search plan*, which controls the application of transition rules, yields a *DPLL($\Gamma + \mathscr{T}$)-strategy*, or *DPLL($\Gamma + \mathscr{T}$)-procedure*. Similar to DPLL($\mathscr{T}$), a search plan for DPLL($\Gamma + \mathscr{T}$) is a depth-first search plan. A standard way to ensure fairness with a depth-first search plan is *iterative deepening*. This section describes first a *DPLL($\Gamma + \mathscr{T}$)-procedure with iterative deepening*, and then a way to use it with SpeculativeIntro to get decision procedures for smooth sets.

Definition 9 For all states $M \parallel F$, for all $C \in \mathit{clauses}(F)$, for all implied literals $l_C \in \mathit{lits}(M)$, and for all decided literals $l \in \mathit{lits}(M)$, the *inference depth* is given by

- $\mathit{inf Depth}(C) = \begin{cases} 0 & \text{if } C \in F_0, \\ n+1 & \text{if } C = \mathsf{C}(\gamma) \text{ and} \\ & n = \mathit{max}\,\{\mathit{inf Depth}(D) \mid D \in \mathsf{P}(\gamma)\} \text{ in a Deduce step,} \end{cases}$
- $\mathit{inf Depth}(l_C) = \mathit{inf Depth}(C)$ and
- $\mathit{inf Depth}(l) = \mathit{min}\,\{\mathit{inf Depth}(D) \mid D \in \mathit{clauses}(F),\ l \in D\}$.

Informally, the inference depth of a clause indicates the depth of the inference tree that produced it; the inference depth of an implied literal is the inference depth of the clause that implied it; and the inference depth of a decided literal is the minimum inference depth of a clause that includes it.

In order to have a DPLL($\Gamma + \mathscr{T}$)-procedure with iterative deepening, both rules susceptible of yielding infinitely many steps need to be bounded:

Definition 10 DPLL($\Gamma + \mathscr{T}$) is $\langle k_d, k_u \rangle$-*bounded*, for $k_d, k_u > 0$, if Deduce is restricted to premises C with $\mathit{inf Depth}(C) < k_d$, and SpeculativeIntro can be applied at most k_u times.

This notion leads to termination:

Theorem 5 *$\langle k_d, k_u \rangle$-bounded DPLL($\Gamma + \mathscr{T}$) is guaranteed to terminate for all initial sets of clauses $S = \mathscr{R} \uplus P$.*

Proof By Definition 10, there are only finitely many applications of Deduce and SpeculativeIntro. The other DPLL($\mathscr{T}$) transition rules are known to terminate (e.g., [55], cf. Theorems 2.10 and 3.7). □

Definition 11 DPLL($\Gamma + \mathscr{T}$) is *stuck* at k_d in state $M \parallel F$ if the only applicable transitions are SpeculativeIntro transitions and Deduce transitions involving premises C with $\mathit{inf Depth}(C) \geq k_d$.

A *DPLL($\Gamma + \mathscr{T}$)-procedure with iterative deepening*, abbreviated *ID-DPLL($\Gamma + \mathscr{T}$)-procedure*, is a DPLL($\Gamma + \mathscr{T}$)-procedure where DPLL($\Gamma + \mathscr{T}$) is $\langle k_d, k_u \rangle$-bounded, and k_d and k_u are increased whenever DPLL($\Gamma + \mathscr{T}$) gets stuck. The following example shows how fairness is not obvious without iterative deepening:

Example 1 Let Γ be an inference system with resolution and let F_0 include the following clauses:

(1) $\neg p(x, y) \vee p(f(x), f(y)) \vee p(g(x), g(y))$,
(2) $p(a, b)$,
(3) $g(x) \not\simeq x$,
(4) $g(c) \simeq c \vee g(d) \simeq d$.

Initially, Γ sees clauses (1) and (3), because they are in $\mathit{ngclauses}(F)$, while $\mathit{lits}_{\mathscr{R}}(M)$ is empty. If Decide adds $p(a, b)$ to M, Γ sees also (2) and may generate

$$p(f(a), f(b)) \vee p(g(a), g(b))$$

from (1) and (2) by resolution. If Decide adds $p(f(a), f(b))$ to M, and Γ generates

$$p(f(f(a)), f(f(b))) \vee p(g(f(a)), g(f(b))),$$

this alternation of decision and resolution steps may yield an infinite unfair derivation that does not detect the unsatisfiability of F_0. Iterative deepening prevents this kind of behavior: when the depth of the clauses generated by resolution reaches the bound, further such steps are forbidden and the system is forced to consider steps with clauses of lower depth. When Decide adds to M first $g(c) \simeq c$ and then $g(d) \simeq d$ and each yields $\Box$ by resolution with $g(x) \not\simeq x$, inconsistency is detected.

Let S be a smooth set, and let U denote a sequence of "speculative axioms," in the signature of S, that are introduced by SpeculativeIntro. In order to get a decision procedure, one needs to show that for some sequence U, there exist bounds k_d and k_u, such that $\langle k_d, k_u \rangle$-bounded DPLL($\Gamma + \mathscr{T}$) is guaranteed to terminate in the *unsat* state, whenever S is unsatisfiable, and in a state $M \parallel F$ such that DPLL($\Gamma + \mathscr{T}$) is not *stuck* at k_d, whenever S is satisfiable; note that this means that $M \parallel F$ is saturated. The second example illustrates this idea:

Example 2 Let $\mathscr{R}$ 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 SpeculativeIntro adds

$$\lceil f(x) \simeq x \rceil \triangleright f(x) \simeq x,$$

the monotonicity axiom is rewritten to a tautology and $a \sqsubseteq f(c)$ is also rewritten. Note that $\lceil f(x) \simeq x \rceil$ is a decision literal, and $level(\lceil f(x) \simeq x \rceil) = 1$. Thus, the rewriting steps only disable the monotonicity axiom and $a \sqsubseteq f(c)$, whose scope level is 0, and add $\lceil f(x) \simeq x \rceil \triangleright a \sqsubseteq c$ to F. Resolution generates the conflict clause $\lceil f(x) \simeq x \rceil \triangleright \Box$. In conflict resolution mode, the literal $\neg\lceil f(x) \simeq x \rceil$ is added to M, preventing DPLL($\Gamma + \mathscr{T}$) from guessing $f(x) \simeq x$ again. Next, if SpeculativeIntro 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($\Gamma + \mathscr{T}$) reaches a saturated state, and satisfiability is detected.

The third example shows a case where PropagateEq plays the key rôle:

Example 3 Let Γ have hyperresolution, superposition and simplification, $\mathscr{T}$ be the theory of linear integer arithmetic, $\mathscr{R}$ be

$$\{\neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq z) \vee x \sqsubseteq z\}$$

and P be

$$\{a \sqsubseteq b_1,\ b_2 \sqsubseteq c,\ \neg(a \sqsubseteq c), b_1 \leq b_2, b_1 > b_2 - 1\}.$$

UnitPropagate adds the literals of P to M. In the model $\mathsf{model}_{LA}(M)$ maintained by the linear arithmetic solver, $\mathsf{model}_{LA}(M)(b_1) = \mathsf{model}_{LA}(M)(b_2)$. Thus, PropagateEq *guesses* the equation $b_1 \simeq b_2$. Say $b_2 \succ b_1$ in the ordering $\succ$ of Γ: 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 the transitivity axiom, so that an inconsistency is detected. DPLL($\Gamma + \mathscr{T}$) 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.

6 Decision Procedures for Axiomatizations of Type Systems

In this section we study specific theories of interest for software verification and we obtain decision procedures for them.

Definition 12 A structure Φ is *essentially finite* with respect to a function symbol f if $\mathsf{range}(\Phi(f))$ is finite.

Essential finiteness is weaker than finiteness, because it admits an infinite domain provided $\mathsf{range}(\Phi(f))$ is finite.

Theorem 6 *If Φ is an essentially finite structure with respect to a monadic function symbol f, then there exist $k_1, k_2 \geq 0$, $k_1 \neq k_2$, 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 w in the lasso of the f-chain starting at v, $\Phi(f)^n(w) = w$, when $n = \mathsf{ls}(\Phi, f, v)$. Also, for all multiples of the lasso, that is, for all $n = j \cdot \mathsf{ls}(\Phi, f, v)$ for some integer $j > 0$, $\Phi(f)^n(w) = w$. Let $q = \mathsf{max}\{\mathsf{pr}(\Phi, f, v) \mid v \in \mathsf{range}(\Phi(f))\} + 1$ and $n = \mathsf{lcm}\{\mathsf{ls}(\Phi, f, v) \mid v \in \mathsf{range}(\Phi(f))\}$, where lcm abbreviates least common multiple. We claim that $\Phi \models f^{q+n}(x) \simeq f^q(x)$, that is, $k_1 = q + n$ and $k_2 = q$. By way of contradiction, assume that for some $v \in |\Phi|$, $\Phi(f)^{q+n}(v) \neq \Phi(f)^q(v)$. Take the f-chain starting at v: $\Phi(f)^q(v)$ is in the lasso of this chain, because $q > \mathsf{pr}(\Phi, f, v)$. Since n is a multiple of $\mathsf{ls}(\Phi, f, v)$, we have $\Phi(f)^{q+n}(v) = \Phi(f)^n(\Phi(f)^q(v)) = \Phi(f)^q(v)$, a contradiction. □

Example 4 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, $q = max\{2, 0\} + 1 = 3$, $n = lcm\{3, 4\} = 12$, $k_1 = q + n = 15$ and $k_2 = q = 3$, and $\Phi \models f^{15}(x) \simeq f^3(x)$.

To identify classes of problems for which an ID-DPLL($\Gamma + \mathscr{T}$)-procedure is a decision procedure, we focus on theories $\mathscr{R}$ that satisfy the following property:

Definition 13 A presentation $\mathscr{R}$ is *essentially finite* if its signature contains a single monadic function symbol f, and for all sets P of ground $\mathscr{R}$-clauses, such that $\mathscr{R} \uplus P$ is satisfiable, $\mathscr{R} \uplus P$ has an essentially finite model Φ with respect to f.

We show that ID-DPLL($\Gamma + \mathscr{T}$) is a decision procedure for essentially finite theories if the number of literals in clauses is bounded:

Theorem 7 *Let $\mathscr{R}$ be an essentially finite presentation. Consider an ID-DPLL($\Gamma + \mathscr{T}$)-procedure where every* SpeculativeIntro *transition adds an equation $f^j(x) \simeq f^k(x)$ with $j > k$, for increasing values of j and k. If there exists an n such that no clause generated by DPLL($\Gamma + \mathscr{T}$) contains more than n literals, ID-DPLL($\Gamma + \mathscr{T}$) is a decision procedure for the satisfiability modulo $\mathscr{T}$ of $\mathscr{T}$-smooth problems $\mathscr{R} \uplus P$.*

Proof If $\mathscr{R} \uplus P$ is unsatisfiable, then, by refutational completeness, DPLL($\Gamma + \mathscr{T}$) will reach the state *unsat* when k_d becomes large enough. If $\mathscr{R} \uplus P$ is satisfiable, it has an essentially finite model Φ, because $\mathscr{R}$ is essentially finite. Choose k_u large enough that the axiom $f^{k_1}(x) \simeq f^{k_2}(x)$ satisfied by Φ according to Theorem 6 is added by SpeculativeIntro. We need to prove that if k_d is large enough, DPLL($\Gamma + \mathscr{T}$) 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 chosen k_u. Say that $k_1 > k_2$: the axiom $f^{k_1}(x) \simeq f^{k_2}(x)$ is applied as a rewrite rule $f^{k_1}(x) \to f^{k_2}(x)$ to simplify[5] all clauses that contain a term $f^k(t)$ with $k > k_1$. This guarantees that no such term will be kept and that the depth of terms in clauses is bounded. Since the number of literals in clauses is also bounded by the hypothesis that no clause can contain more than n literals, only a finite number of clauses can be derived for unbounded k_d. Thus, DPLL($\Gamma + \mathscr{T}$) will halt without getting stuck and will detect satisfiability. □

From now on, unless otherwise stated, Γ is superposition with negative selection, hyperresolution, factoring and simplification.

Lemma 2 *If $\mathscr{R}$ is Horn, the number of literals in clauses generated by DPLL($\Gamma + \mathscr{T}$) from a $\mathscr{T}$-smooth $\mathscr{R} \uplus P$ is bounded.*

Proof In the Horn case, superposition is unit superposition, which does not increase the number of literals, and hyperresolution only generates positive unit clauses. □

[5] Of course, this assumes that Γ features simplification.

If $\mathscr{R}$ is a set of non-equational clauses with no more than two literals each, and Γ is resolution, factoring and 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 14 A clause $C = \neg l_1 \vee \ldots \vee \neg l_n \vee l_{n+1} \vee \ldots \vee l_{n+q}$ is *ground-preserving* if

$$\bigcup_{j=n+1}^{n+q} Var(l_j) \subseteq \bigcup_{j=1}^{n} Var(l_j).$$

A set of clauses is *ground-preserving* if all its clauses are.

In a ground-preserving set the only positive clauses are ground.[6]

Lemma 3 *If $\mathscr{R}$ is essentially finite and ground-preserving, and every* SpeculativeIntro *transition adds an equation $f^j(x) \simeq f^k(x)$ with $j > k$, for increasing values of j and k, $DPLL(\Gamma + \mathscr{T})$ generates finitely many clauses from a $\mathscr{T}$-smooth $\mathscr{R} \uplus P$.*

Proof Hyperresolution only generates positive ground clauses, because all variables get instantiated by resolving the negative literals with positive clauses. Superposition with negative selection superposes a ground positive clause into a ground-preserving clause, which generates either a ground clause, or a non-ground ground-preserving clause with no more variable positions than its non-ground parent. It follows that superposition creates no new non-ground term, and only finitely many non-ground ground-preserving clauses can be derived. Since term depth is limited by simplification by $f^{k_1}(x) \to f^{k_2}(x)$, only finitely many ground clauses can be generated. □

Next, we consider some specific theories relevant to the axiomatization of type systems in programming languages. Given the axioms

$$\text{Reflexivity} \quad x \sqsubseteq x \tag{1}$$

$$\text{Transitivity} \quad \neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq z) \vee x \sqsubseteq z \tag{2}$$

$$\text{Anti-Symmetry} \quad \neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq x) \vee x \simeq y \tag{3}$$

$$\text{Monotonicity} \quad \neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y) \tag{4}$$

$$\text{Tree-Property} \quad \neg(z \sqsubseteq x) \vee \neg(z \sqsubseteq y) \vee x \sqsubseteq y \vee y \sqsubseteq x \tag{5}$$

{(1), (2), (3)} presents a poset (partially ordered set), $\mathsf{MI} = \{(1), (2), (3), (4)\}$ a type system with *multiple inheritance*, and $\mathsf{SI} = \mathsf{MI} \uplus \{(5)\}$ a type system with *single inheritance*, where $\sqsubseteq$ is the subtype relationship and f is a type constructor. MI and SI are essentially finite, because they satisfy a stronger property:

Definition 15 $\mathscr{R}$ has the *finite model property*, if for all sets P of ground $\mathscr{R}$-clauses, such that $\mathscr{R} \uplus P$ is satisfiable, $\mathscr{R} \uplus P$ has a model Φ with finite $|\Phi|$.

[6] Definition 14 is a weakening of that of positive variable dominated clause of [22] (cf. Definition 3.18), and it is dual to that of ground-preserving clause of [47], which required that negative literals do not contain variables that do not appear in positive ones. Our definition is for a positive strategy in the non-Horn case, hence forward reasoning, whereas that of [47] was for linear input proofs in the Horn case, hence backward reasoning.

Theorem 8 SI *has the finite model property hence it is essentially finite.*

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 $V_P \cup \{v_m\}$, where v_m is an element not in V_P. Then, we define $\Phi'(\sqsubseteq)(v_1, v_2)$ as:

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

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

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

Let dom_f, the relevant domain of f, be the set $\{\Phi(t) \mid f(t) \in T_P\} \cup \{v_m\}$. 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 $v_m \in \mathsf{dom}_f$, v_m is the maximal element of $|\Phi'|$, and $\Phi'(\sqsubseteq)$ satisfies the Tree-Property, which ensures uniqueness of the image. Moreover, $\Phi'(f)$ is monotonic with respect to $\Phi'(\sqsubseteq)$. □

Definition 16 Let $\langle A, \sqsubseteq\rangle$ be a poset. The *Dedekind–MacNeille completion* [50] 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 greatest and least lower bound, and
- B is finite if A is finite. Actually, B is a subset of 2^A.

Theorem 9 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 Tree-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:

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

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 [23] 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 ID-DPLL($\Gamma + \mathscr{T}$) is a decision procedure for MI and SI.

Theorem 10 *An ID-DPLL($\Gamma + \mathscr{T}$)-procedure where every* SpeculativeIntro *transition adds an equation $f^j(x) \simeq f^k(x)$ with $j > k$, for increasing values of j and k, is a decision procedure for the satisfiability modulo $\mathscr{T}$ of $\mathscr{T}$-smooth problems* MI $\uplus$ *P.*

Proof It follows from Theorem 7 and Lemma 2, because MI is essentially finite and Horn. □

Theorem 11 *An ID-DPLL($\Gamma + \mathscr{T}$)-procedure where every* SpeculativeIntro *transition adds an equation $f^j(x) \simeq f^k(x)$ with $j > k$, for increasing values of j and k, is a decision procedure for the satisfiability modulo $\mathscr{T}$ of $\mathscr{T}$-smooth problems* SI $\uplus$ *P.*

Proof Since SI is essentially finite and ground-preserving, except for Reflexivity, it follows from Theorem 7 and Lemma 3, provided Reflexivity does not affect the result of Lemma 3. This is the case, since an hyperresolution involving Reflexivity generates either a tautology or a subsumed clause or a ground clause. □

In Spec# [7], the axiomatization of the type system also includes TR $= \{g(x) \not\simeq null,\ h(g(x)) \simeq x\}$, where g represents the *type representative* of some type. The first axiom states that the representative is never the constant *null*, which means *null* has no pre-image, hence g is not surjective. The second axiom states that g has a left inverse, hence it is injective. It is well-known that a set with an injective but not surjective function is infinite (e.g., Lemma 1 in [24]), so that any model of TR is infinite.

Theorem 12 *An ID-DPLL($\Gamma + \mathscr{T}$)-procedure where every* SpeculativeIntro *transition adds an equation $f^j(x) \simeq f^k(x)$ with $j > k$, for increasing values of j and k, is a decision procedure for the satisfiability modulo $\mathscr{T}$ of $\mathscr{T}$-smooth problems* MI $\uplus$ TR $\uplus$ *P and* SI $\uplus$ TR $\uplus$ *P.*

Proof Superposition applied to an axiom in TR and a ground equation generates a ground equation smaller than its ground parent in the $\succ$ ordering (e.g., under a precedence $g \succ h \succ null$), so that Γ terminates on TR-satisfiability problems. Since MI (or SI) and TR are disjoint and variable inactive, Γ terminates also on satisfiability problems in MI $\uplus$ TR (or SI $\uplus$ TR) by Theorem 2. Thus, the combination with TR does not change that only finitely many clauses can be generated. The claim follows from Theorem 10 and this observation for problems MI $\uplus$ TR $\uplus$ P, and from Theorem 11 and this observation for problems SI $\uplus$ TR $\uplus$ P. □

7 Discussion

The DPLL($\Gamma + \mathcal{T}$) system integrates DPLL($\mathcal{T}$) with a first-order engine Γ, to combine the strengths of DPLL and efficient solvers for special theories, such as

linear arithmetic, with those of superposition and resolution. DPLL($\mathcal{T}$)-based SMT-solvers and general theorem provers grew independently for several years. The increasing recognition that their features are complementary, and necessary to solve frontier problems in fields such as program verification, is leading to study their interaction.

The rewrite-based approach to satisfiability procedures developed in [3–5, 15, 18] was concerned with using first-order engines as decision procedures for satisfiability problems. In [14, 16] it was generalized from satisfiability problems, given by sets of ground unit clauses, to decision problems given by sets of ground clauses. The first-order engine alone was used as decision procedure, with no integration with an SMT-solver. The two-stage method of [13, 17] lets a first-order engine and an SMT-solver cooperate, including allowing both a union $\mathscr{R}$ of variable-inactive axiomatized theories, and a union $\mathscr{T}$ of Nelson-Oppen built-in theories: the first-order engine was applied as a pre-processor to compile $\mathscr{R}$ and reduce it to a theory that DPLL($\mathcal{T}$) alone could handle. Thus, the two reasoners were applied in sequence. The systems in [1, 46] explored embedding a $\mathcal{T}$-solver for linear arithmetic into a superposition-based theorem prover, while the study of *iterated schemata* in [2] offers another perspective on the integration of propositional and arithmetical reasoning.

In DPLL($\Gamma + \mathcal{T}$) the first-order engine Γ is tightly integrated within DPLL($\mathcal{T}$), resulting in one single system. This is a main difference with respect to the two-stage approach of [13, 17]. DPLL($\Gamma + \mathcal{T}$) and systems such as those in [1, 46] can be considered symmetric: in DPLL($\Gamma + \mathcal{T}$) the superposition-based engine Γ is a satellite of DPLL($\mathcal{T}$); in [1, 46] the $\mathcal{T}$-solver is a satellite of the superposition-based engine. A first version of DPLL($\Gamma + \mathcal{T}$) appeared in [28]. It was called DPLL(Γ), because it was known to be refutationally complete only in the case where the background theory $\mathscr{T}$ is empty. A first contribution of this article was to advance the DPLL($\Gamma + \mathcal{T}$) approach by giving conditions under which it is refutationally complete when both $\mathscr{R}$ and $\mathscr{T}$ are not empty.

Combination of theories is of paramount importance to reason about software. In previous work, it was known how to combine built-in theories, according to the equality sharing method pioneered by Nelson and Oppen [53], and studied since then by many authors (e.g., [37, 54, 65] for some of the most recent extensions). This style of combination requires one to embed in the prover a decision procedure for each theory of interest. While decision procedures are available for several theories, it might not be the case for each and every group of axioms that may appear in program checking problems. This is a main reason why we need an axiomatized theory $\mathscr{R}$ and Γ to reason about it. A second main contribution of this article was to show how to let combination of built-in theories à la Nelson-Oppen, and union of axiomatized theories under variable inactivity, coexist and work together in DPLL($\Gamma + \mathscr{T}$).

We presented a new DPLL($\Gamma + \mathscr{T}$) system that combines DPLL($\Gamma + \mathscr{T}$) with speculative inferences. The purpose is to enforce termination by introducing additional axioms as hypotheses. This idea was inspired by the "unsound theorem proving" concept of [49]. The additional axioms may cause unsoundness, by making unsatisfiable what was a satisfiable set. We provided a mechanism for the prover to detect any unsoundness introduced by the added axioms and recover from it. This mechanism is based on the backtracking scheme that is native of a DPLL search: an inconsistency due to a speculative inference is an *"unnatural failure"* that the prover

treats like a *"natural failure"* (a proper inconsistency) by backtracking. Furthermore, it keeps memory of the failure to avoid repeating it. An idea of speculative inferences may be implicit in bottom-up model generation approaches [9]. In those contexts the speculation consists of trying in turn each case in a case analysis. In our method, the speculative inferences assert additional clauses, on top of the native case analysis of DPLL on the input clauses. Considering each of the two horns of a case analysis can be seen either as an inference step or as a search step on existing data. Our speculative inferences are more like guessing additional features that a model may satisfy.

DPLL($\Gamma + \mathscr{T}$) equipped with an iterative deepening search plan forms an *ID-DPLL($\Gamma + \mathscr{T}$)-procedure*. We showed that ID-DPLL($\Gamma + \mathscr{T}$) with speculative inferences is a decision procedure for theories that axiomatize type systems relevant to program checking. Their crucial feature is that they are *essentially finite*: they have one unary function symbol whose range is finite. However, we gave examples where ID-DPLL($\Gamma + \mathscr{T}$) is a decision procedure also when more function symbols are involved via combination of theories. Another way to approach the axiomatizations in Section 6 is *locality*, proposed for Horn theories [40], and then extended beyond the Horn case and developed in [8, 43, 44, 63, 64]. In a local theory, validity of a conjecture can be decided by considering only finitely many of its ground instances. We emphasize that DPLL($\Gamma + \mathscr{T}$) yields a decision procedure for the axiomatizations in Section 6 united with an arbitrary built-in $\mathscr{T}$, provided the problem $\mathscr{T} \uplus \mathscr{R} \uplus P$ is $\mathscr{T}$-smooth. In applications, there is no guarantee that all relevant instances of $\mathscr{T} \uplus \mathscr{R}$ will be local. Thus, speculative inferences and locality can be considered complementary.

There are several directions for future work. One is to extend the approach to more presentations, including cases where the signature of $\mathscr{R}$ features also non-monadic function symbols. For example, consider the axiom $y \sqsubseteq x \wedge u \sqsubseteq z \Rightarrow map(x, u) \sqsubseteq map(y, z)$, where $\sqsubseteq$ is a subtype relation, and $map(x, u)$ represents the type of maps from type x to type u. If $y \sqsubseteq x$, a value of type y can be used whenever a value of type x is used. For maps $f \in map(x, u)$ and $g \in map(y, z)$ this is the case if $y \sqsubseteq x$, which means f can take whatever g takes, and $u \sqsubseteq z$, which means whatever f yields is within what g yields. Such an axiom with a dyadic function symbol may be useful for an axiomatization of maps. Another open issue is the duplication of reasoning on ground unit equational clauses in DPLL($\Gamma + \mathcal{T}$), due to the fact that they 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 [38, 62], and pass the resulting canonical system to Γ, would not solve the problem, because this solution is not *incremental*, as the addition of a single ground equation requires recomputing the canonical system. It would be ideal to automate the choice of clauses to be added by SpeculativeIntro. However, this manual component of DPLL($\Gamma + \mathcal{T}$) is at a higher level of abstraction than triggering, and it is certainly not heavier.

Another topic for future investigation is how to improve the capability of the system to discover unsatisfiability due to the lack of finite models. By detecting the generation of a cardinality constraint by Γ, DPLL($\Gamma + \mathscr{T}$) can discover that an axiomatized theory $\mathscr{R}$ is not variable-inactive and not stably infinite. In other words, it can discover the lack of infinite models. On the other hand, it does not have a

general way to discover the lack of finite models, or that there are only models with the "wrong" cardinality: for example, $\mathscr{R}$ only features models with a certain finite cardinality, when $\mathscr{T}$ requires a different finite cardinality.

The class of formulae that can be decided by DPLL($\Gamma + \mathcal{T}$) includes axiomatizations of type systems, used in tools such as ESC/Java [36] and Spec# [7], which represents significant evidence of the relevance of this work to applications.

Acknowledgements Part of this research initiated during a visit of the first author with the Software Reliability Group of Microsoft Research in Redmond. We thank the anonymous reviewers whose suggestions allowed us to improve an earlier version of this article.

References

1. Ernst, E., Kruglov, E., Weidenbach, C.: Superposition modulo linear arithmetic SUP(LA). In: Ghilardi, S., Sebastiani, R. (eds.) Proceedings of the Seventh Symposium on Frontiers of Combining Systems (FroCoS). Lecture Notes in Artificial Intelligence, vol. 5749, pp. 84–99. Springer (2009)
2. Vincent, V., Caferra, R., Peltier, N.: A decidable class of nested iterated schemata. In: Giesl, J., Hähnle, R. (eds.) Proceedings of the Fifth International Joint Conference on Automated Reasoning (IJCAR). Lecture Notes in Artificial Intelligence, vol. 6173, pp. 293–308. Springer (2010)
3. Armando, A., Bonacina, M.P., Ranise, S., Schulz, S.: On a rewriting approach to satisfiability procedures: extension, combination of theories and an experimental appraisal. In: Gramlich, B. (ed.) Proceedings of the Fifth Workshop on Frontiers of Combining Systems (FroCoS). Lecture Notes in Artificial Intelligence, vol. 3717, pp. 65–80. Springer (2005)
4. Armando, A., Bonacina, M.P., Ranise, S., Schulz, S.: New results on rewrite-based satisfiability procedures. ACM Trans. Comput. Log. **10**(1), 129–179 (2009)
5. Armando, A., Ranise, S., Rusinowitch, M.: A rewriting approach to satisfiability procedures. Inf. Comput. **183**(2), 140–164 (2003)
6. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. **4**(3), 217–247 (1994)
7. Barnett, M., Leino, K.R.M., Schulte, W.: The Spec# programming system: an overview. In: Barthe, G., Burdy, L., Huisman, M., Lanet, J.-L., Muntean, T. (eds.) Proceedings of the Workshop on Construction and Analysis of Safe, Secure, and Interoperable Smart Devices (CASSIS 2004). Lecture Notes in Computer Science, vol. 3362, pp. 49–69. Springer (2005)
8. Basin, D.A., Ganzinger, H.: Automated complexity analysis based on ordered resolution. J. ACM **48**(1), 70–109 (2001)
9. Baumgartner, P., Schmidt, R.A.: Blocking and other enhancements for bottom-up model generation methods. In: Furbach, U., Shankar, N. (eds.) Proceedings of the Third International Joint Conference on Automated Reasoning (IJCAR). Lecture Notes in Artificial Intelligence, vol. 4130, pp. 125–139. Springer (2006)
10. Bonacina, M.P.: A taxonomy of theorem-proving strategies. In: Wooldridge, M.J., Veloso, M. (eds.) Artificial Intelligence Today—Recent Trends and Developments. Lecture Notes in Artificial Intelligence, vol. 1600, pp. 43–84. Springer (1999)
11. Bonacina, M.P.: On theorem proving for program checking—historical perspective and recent developments. In: Fernandez, M. (ed.) Proceedings of the Twelfth International Symposium on Principles and Practice of Declarative Programming (PPDP), pp. 1–11. ACM Press (2010)
12. Bonacina, M.P., Dershowitz, N.: Abstract canonical inference. ACM Trans. Comput. Log. **8**(1), 180–208 (2007)
13. Bonacina, M.P., Echenim, M.: $\mathcal{T}$-decision by decomposition. In: Pfenning, F. (ed.) Proceedings of the Twenty-First Conference on Automated Deduction (CADE). Lecture Notes in Artificial Intelligence, vol. 4603, pp. 199–214. Springer (2007)
14. Bonacina, M.P., Echenim, M.: Rewrite-based decision procedures. In: Archer, M., de la Tour, T.B., Munoz, C. (eds.) Proceedings of the Sixth Workshop on Strategies in Automated Deduction (STRATEGIES), Federated Logic Conference 2006. Electronic Notes in Theoretical Computer Science, vol. 174(11), pp. 27–45. Elsevier (2007)

15. Bonacina, M.P., Echenim, M.: Rewrite-based satisfiability procedures for recursive data structures. In: Cook, B., Sebastiani, R. (eds.) Proceedings of the Fourth Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR), Federated Logic Conference 2006. Electronic Notes in Theoretical Computer Science, vol. 174(8), pp. 55–70. Elsevier (2007)
16. Bonacina, M.P., Echenim, M.: On variable-inactivity and polynomial T-satisfiability procedures. J. Log. Comput. **18**(1), 77–96 (2008)
17. Bonacina, M.P., Echenim, M.: Theory decision by decomposition. J. Symb. Comput. **45**(2), 229–260 (2010)
18. Bonacina, M.P., Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Decidability and undecidability results for Nelson–Oppen and rewrite-based decision procedures. In: Furbach, U., Shankar, N. (eds.) Proceedings of the Third International Joint Conference on Automated Reasoning (IJCAR). Lecture Notes in Artificial Intelligence, vol. 4130, pp. 513–527. Springer (2006)
19. Bonacina, M.P., Hsiang, J.: Towards a foundation of completion procedures as semidecision procedures. Theoret. Comput. Sci. **146**, 199–242 (1995)
20. Bonacina, M.P., Lynch, C.A., de Moura, L.: On deciding satisfiability by DPLL($\Gamma + \mathcal{T}$) and unsound theorem proving. In: Schmidt, R. (ed.) Proceedings of the Twenty-Second Conference on Automated Deduction (CADE). Lecture Notes in Artificial Intelligence, vol. 5663, pp. 35–50. Springer (2009)
21. Bradley, A.R., Manna, Z., Sipma, H.B.: What's decidable about arrays? In: Emerson, E.A., Namjoshi, K.S. (eds.) Proceedings of the Seventh Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI). Lecture Notes in Computer Science, vol. 3855, pp. 427–442. Springer (2006)
22. Caferra, R., Leitsch, A., Peltier, N.: Automated Model Building. Kluwer Academic Publishers, Amsterdam (2004)
23. Cantone, D., Zarba, C.G.: A decision procedure for monotone functions over bounded and complete lattices. In: de Swart, H. (ed.) Proc. TARSKI II. Lecture Notes in Artificial Intelligence, vol. 4342, pp. 318–333. Springer (2006)
24. Claessen, K., Lillieström, A.: Automated inference of finite unsatisfiability. In: Schmidt, R. (ed.) Proceedings of the Twenty-Second Conference on Automated Deduction (CADE). Lecture Notes in Artificial Intelligence, vol. 5663, pp. 388–403. Springer (2009)
25. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
26. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**, 201–215 (1960)
27. de Moura, L., Bjørner, N.: Efficient E-matching for SMT-solvers. In: Pfenning, F. (ed.) Proceedings of the Twenty-First Conference on Automated Deduction (CADE). Lecture Notes in Artificial Intelligence, vol. 4603, pp. 183–198. Springer (2007)
28. de Moura, L., Bjørner, N.: Engineering DPLL(T) + saturation. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Proceedings of the Fourth International Joint Conference on Automated Reasoning (IJCAR). Lecture Notes in Artificial Intelligence, vol. 5195, pp. 475–490. Springer (2008)
29. de Moura, L., Bjørner, N.: Model-based theory combination. In: Krstić, S., Oliveras, A. (eds) Proceedings of the Fifth Workshop on Satisfiability Modulo Theories (SMT), Conference on Automated Verification 2007. Electronic Notes in Theoretical Computer Science, vol. 198(2), pp. 37–49. Elsevier (2008)
30. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Proceedings of the Fourteenth Conference on Tools and algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 4963, pp. 337–340 (Springer).
31. Dershowitz, M.: Orderings for term-rewriting systems. Theore. Comput. Sci. **17**(3), 279–301 (1982)
32. Dershowitz, N.: A maximal-literal unit strategy for Horn clauses. In: Kaplan, S., Okada, M. (eds.) Proceedings of the Second Workshop on Conditional and Typed Term Rewriting Systems (CTRS 1990). Lecture Notes in Computer Science, vol. 516, pp. 14–25. Springer (1991)
33. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Commun. ACM **22**(8), 465–476 (1979)
34. Detlefs, D.L., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM **52**(3), 365–473 (2005)
35. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) Proceedings of the Eighteenth Conference on Automated Verification (CAV). Lecture Notes in Computer Science, vol. 4144, pp. 81–94. Springer (2006)

36. Flanagan, C., Leino, K.R.M., Lillibridge, M., Nelson, G., Saxe, J.B., Stata, R.: Extended static checking for Java. In: Hendren, L.J. (ed.) ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pp. 234–245 (2002)
37. Fontaine, P.: Combinations of theories for decidable fragments of first-order logic. In: Ghilardi, S., Sebastiani, R. (eds.) Proceedings of the Seventh Symposium on Frontiers of Combining Systems (FroCoS). Lecture Notes in Artificial Intelligence, vol. 5749, pp. 263–278. Springer (2009)
38. Gallier, J., Narendran, P., Plaisted, D.A., Raatz, S., Snyder, W.: Finding canonical rewriting systems equivalent to a finite set of ground equations in polynomial time. J. ACM **40**(1), 1–16 (1993)
39. Ge, Y., Barrett, C., Tinelli, C.: Solving quantified verification conditions using satisfiability modulo theories. In: Pfenning, F. (ed.) Proceedings of the Twenty-First Conference on Automated Deduction (CADE). Lecture Notes in Artificial Intelligence, vol. 4603, pp. 167–182. Springer (2007)
40. Givan, R., McAllester, D.A.: Polynomial-time computation via local inference relations. ACM Trans. Comput. Log. **3**(4), 521–541 (2002)
41. Halpern, J.Y.: Presburger arithmetic with unary predicates is π_1^1 complete. J. Symb. Log. **56**, 637–642 (1991)
42. Hsiang, J., Rusinowitch, M.: Proving refutational completeness of theorem proving strategies: the transfinite semantic tree method. J. ACM **38**(3), 559–587 (1991)
43. Ihlemann, C., Jacobs, S., Sofronie-Stokkermans, V.: On local reasoning in verification. In: Ramakrishnan, C.R., Rehof, J. (eds.) Proceedings of the Fourteenth Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 4963, pp. 265–281. Springer (2008)
44. Jacobs, S.: Incremental instance generation in local reasoning. In: Baader, F., Ghilardi, S., Hermann, M., Sattler, U., Sofronie-Stokkermans, V. (eds.) Notes of the First Workshop on Complexity, Expressibility and Decidability (CEDAR). International Joint Conference on Automated Reasoning 2008, pp. 47–62 (2008)
45. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: Leech, J. (ed.) Proceedings of the Conference on Computational Problems in Abstract Algebras, pp. 263–298. Pergamon Press (1970)
46. Korovin, K., Voronkov, A.: Integrating linear arithmetic into superposition calculus. In: Duparc, J., Henzinger, T.A. (eds.) Proceedings of the Sixteenth EACSL Annual Conference on Computer Science Logic (CSL). Lecture Notes in Computer Science, vol. 4646, pp. 223–237. Springer (2007)
47. Kounalis, E., Rusinowitch, M.: On word problems in Horn theories. J. Symb. Comput. **11**(1–2), 113–128 (1991)
48. Lifschitz, V., Morgenstern, L., Plaisted, D.A.: Knowledge representation and classical logic. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, vol. 1, pp. 3–88. Elsevier (2008)
49. Lynch, C.A.: Unsound theorem proving. In: Marcinkowski, J., Tarlecki, A. (eds.) Proceedings of the Thirteenth EACSL Annual Conference on Computer Science Logic (CSL). Lecture Notes in Computer Science, vol. 3210, pp. 473–487. Springer (2004)
50. MacNeille, H.M.: Partially ordered sets. Trans. Am. Math. Soc. **42**, 416–460 (1937)
51. McCune, W.W.: Otter 3.3 Reference Manual. Technical Report ANL/MCS-TM-263, MCS Division, Argonne National Laboratory, Argonne, IL, USA (2003)
52. McPeak, S., Necula, G.C.: Data structure specifications via local equality axioms. In: Etessami, K., Rajamani, S.K. (eds.) Proceedings of the Seventeenth Conference on Automated Verification (CAV). Lecture Notes in Computer Science, vol. 3576, pp. 476–490. Springer (2005)
53. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Trans. Program. Lang. Syst. **1**(2), 245–257 (1979)
54. Nicolini, E., Ringeissen, C., Rusinowitch, M.: Data structures with arithmetic constraints: a non-disjoint combination. In: Ghilardi, S., Sebastiani, R. (eds.) Proceedings of the Seventh Symposium on Frontiers of Combining Systems (FroCoS). Lecture Notes in Artificial Intelligence, vol. 5749, pp. 319–334. Springer (2009)
55. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). J. ACM **53**(6), 937–977 (2006)
56. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. 1, pp. 371–443. Elsevier (2001)

57. Robinson, G., Wos, L.: Paramodulation and theorem-proving in first-order theories with equality. In: Michie, D., Meltzer, R. (eds.) Machine Intelligence, vol. IV, pp. 135–150. Edinburgh University Press (1969)
58. Robinson, J.A.: Automatic deduction with hyper-resolution. Int. J. Comput. Math. **1**, 227–234 (1965)
59. Robinson, J.A.: A machine oriented logic based on the resolution principle. J. ACM **12**(1), 23–41 (1965)
60. Rusinowitch, M.: Theorem-proving with resolution and superposition. J. Symb. Comput. **11**, 21–50 (1991)
61. Sebastiani, R.: Lazy satisfiability modulo theories. J. Sat. Bool. Model. and Comput. **3**, 141–224 (2007)
62. Snyder, W.: A fast algorithm for generating reduced ground rewriting systems from a set of ground equations. J. Symb. Comput. **15**(4), 415-450 (1993)
63. Sofronie-Stokkermans, V.: Hierarchic reasoning in local theory extensions. In: Nieuwenhuis, R. (ed.) Proceedings of the Twentieth Conference on Automated Deduction (CADE). Lecture Notes in Artificial Intelligence, vol. 3632, pp. 219–234. Springer (2005)
64. Sofronie-Stokkermans, V., Ihlemann, C.: Automated reasoning in some local extensions of ordered structures. J. Mult.-Valued Log. Soft Comput. **13**(4–6), 397–414 (2007)
65. Wies, T., Piskac, R., Kuncak, V.: Combining theories with shared set operations. In: Ghilardi, S., Sebastiani, R. (eds.) Proceedings of the Seventh Symposium on Frontiers of Combining Systems (FroCoS). Lecture Notes in Artificial Intelligence, vol. 5749, pp. 366–382. Springer (2009)