On First-Order Model-Based Reasoning

Maria Paola Bonacina[1], Ulrich Furbach[2], and Viorica Sofronie-Stokkermans[2]

[1] Dipartimento di Informatica, Università degli Studi di Verona, Italy
mariapaola.bonacina@univr.it

[2] Fachbereich Informatik, Universität Koblenz-Landau, Germany
furbach@uni-koblenz.de, sofronie@uni-koblenz.de

Dedicated to
José Meseguer,
friend and colleague.

Abstract. Reasoning semantically in first-order logic is notoriously a challenge. This paper surveys a selection of *semantically-guided* or *model-based* methods that aim at meeting aspects of this challenge. For first-order logic we touch upon *resolution-based* methods, *tableaux-based* methods, *DPLL-inspired* methods, and we give a preview of a new method called SGGS, for *Semantically-Guided Goal-Sensitive* reasoning. For first-order theories we highlight *hierarchical* and *locality-based* methods, concluding with the recent *Model-Constructing satisfiability calculus*.

1 Introduction

Traditionally, automated reasoning has centered on *proofs* rather than *models*. However, models are useful for applications, intuitive for users, and the notion that *semantic guidance* would help proof search is almost as old as theorem proving itself. In recent years there has been a surge of *model-based* first-order reasoning methods, inspired in part by the success of model-based solvers for propositional satisfiability (SAT) and satisfiability modulo theories (SMT).

The core procedure of these solvers is the *conflict-driven clause learning* (CDCL) version [52, 62, 88, 60] of the Davis-Putnam-Logemann-Loveland (DPLL) procedure for propositional logic [32]. The original Davis-Putnam (DP) procedure [33] was proposed for first-order logic, and featured propositional, or ground, resolution. The DPLL procedure replaced propositional resolution with *splitting*, initially viewed as breaking disjunctions apart by *case analysis*, to avoid the growth of clauses and the non-determinism of resolution. Later, splitting was understood as *guessing*, or *deciding*, the truth value of a propositional variable, in order *to search for a model* of the given set of clauses. This led to read DPLL as a *model-based* procedure, where all operations are centered around a candidate partial model, called *context*, represented by a sequence, or *trail*, of literals.

DPLL-CDCL brought back propositional resolution as a mechanism to generate *lemmas*, and achieve a better balance between *guessing* and *reasoning*. The model-based character of the procedure became even more pronounced: when the

current candidate model falsifies a clause, this *conflict* is *explained* by a heuristically controlled series of resolution steps, a resolvent is added as lemma, and the candidate partial model is repaired in such a way to remove the conflict, satisfy the lemma, and backjump as far away as possible from the conflict. SMT-solvers integrate in DPLL-CDCL a decision procedure for satisfiability in a theory or combination of theories $\mathcal{T}$: the $\mathcal{T}$-satisfiability procedure raises a *$\mathcal{T}$-conflict* when the candidate partial model is not consistent with $\mathcal{T}$, and generates *$\mathcal{T}$-lemmas* to add theory reasoning to the inference component [7, 34].

While SAT and SMT-solvers offer fast decision procedures, they typically apply to sets of propositional or ground clauses, without quantifiers. Indeed, decidability of the problem and termination of the procedure descend from the fact that the underlying language is the *finite* set of the input atoms.

ATP (Automated Theorem Proving) systems offer theorem-proving strategies that are designed for the far more expressive language of first-order logic, but are only *semi-decision procedures* for validity, as the underlying language, and search space, are *infinite*. This trade-off between *expressivity* and *decidability* is ubiquitous in logic and artificial intelligence. First-order satisfiability is not even semi-decidable, which means that first-order model-building cannot be mechanized in general. Nevertheless, there exist first-order reasoning methods that are *semantically-guided* by a *fixed* interpretation, and even *model-based*, in the sense that the state of a derivation contains a representation of a candidate partial model that evolves with the derivation.

In this survey, we illustrate a necessarily incomplete selection of such methods for first-order logic (Section 2) or first-order theories (Section 3). In each section the treatment approximately goes from syntactic or axiomatic approaches towards more semantic ones, also showing connections with José Meseguer's work. All methods are described in expository style, and the interested reader may find the technical details in the references. Background material is available in previous surveys, such as [67, 68, 18, 59, 69] for theorem-proving strategies, [19] for decision procedures based on theorem-proving strategies or their integration with SMT-solvers, and books such as [70, 17, 76].

2 Model-based Reasoning in First-Order Logic

In this section we cover *semantic resolution*, which represents the early attempts at injecting semantics in resolution; *hypertableaux*, which illustrates model-based reasoning in tableaux, with applications to fault diagnosis and description logics; the *model-evolution calculus*, which lifts DPLL to first-order logic, and a new method called *SGGS*, for *Semantically-Guided Goal-Sensitive* reasoning, which realizes a first-order CDCL mechanism.

2.1 Semantic Resolution

Soon after the seminal article by Alan Robinson introducing the resolution principle [75], James R. Slagle presented *semantic resolution* in [79]. Let S be the

finite set of first-order clauses to be refuted. Slagle's core idea was to use a given Herbrand interpretation I to avoid generating resolvents that are true in I, since expanding a consistent set should not lead to a refutation. The following example from [31] illustrates the concept in propositional logic:

Example 1. Given $S = \{\neg A_1 \vee \neg A_2 \vee A_3,\ A_1 \vee A_3,\ A_2 \vee A_3\}$, let I be all-negative, that is, $I = \{\neg A_1, \neg A_2, \neg A_3\}$. Resolution between $\neg A_1 \vee \neg A_2 \vee A_3$ and $A_1 \vee A_3$ generates $\neg A_2 \vee A_3$, after merging identical literals. Similarly, resolution between $\neg A_1 \vee \neg A_2 \vee A_3$ and $A_2 \vee A_3$ generates $\neg A_1 \vee A_3$. However, these two resolvents are true in I. Semantic resolution prevents generating such resolvents, and uses all three clauses to generate only A_3, which is false in I.

Formally, say that we have a clause N, called *nucleus*, and clauses $E_1, \ldots, E_q$, with $q \geq 1$, called *electrons*, such that the electrons are *false* in I. Then, if there is a series of clauses $R_1,\ R_2, \ldots, R_q,\ R_{q+1}$, where R_1 is N, R_{i+1} is a resolvent of R_i and E_i, for $i = 1, \ldots, q$, and R_{q+1} is *false* in I, semantic resolution generates only R_{q+1}. The intuition is that electrons are used to resolve away literals in the nucleus until a clause false in I is generated.

Example 2. In the above example, $\neg A_1 \vee \neg A_2 \vee A_3$ is the nucleus N, and $A_1 \vee A_3$ and $A_2 \vee A_3$ are the electrons E_1 and E_2, respectively. Resolving N and E_1 gives $\neg A_2 \vee A_3$, and resolving the latter with E_2 yields A_3: only A_3 is retained, while the intermediate resolvent $\neg A_2 \vee A_3$ is not.

Semantic resolution can be further restricted by assuming a precedence $>$ on predicate symbols, and stipulating that in each electron the predicate symbol of the literal resolved upon must be maximal in the precedence. The following example also from [31] is in first-order logic:

Example 3. For $S = \{Q(x) \vee Q(a) \vee \neg R(y) \vee \neg R(b) \vee S(c),\ \neg Q(z) \vee \neg Q(a),\ R(b) \vee S(c)\}$, let I be $\{Q(a), Q(b), Q(c), \neg R(a), \neg R(b), \neg R(c), \neg S(a), \neg S(b), \neg S(c)\}$, so that $I \not\models \neg Q(z) \vee \neg Q(a)$ and $I \not\models R(b) \vee S(c)$. Assume the precedence $Q > R > S$. Thus, $Q(x) \vee Q(a) \vee \neg R(y) \vee \neg R(b) \vee S(c)$ is the nucleus N, and $\neg Q(z) \vee \neg Q(a)$ and $R(b) \vee S(c)$ are the electrons E_1 and E_2, respectively. Resolution between N and E_1 on the Q-literals produces $\neg R(y) \vee \neg R(b) \vee S(c)$, which is not false in I, and therefore it is not kept. Note that this resolution step is a binary resolution step between a factor of N and a factor of E_1. Resolution between $\neg R(y) \vee \neg R(b) \vee S(c)$ and E_2 on the R-literals yields $S(c)$. This second resolution step is a binary resolution between a factor of $\neg R(y) \vee \neg R(b) \vee S(c)$ and E_2. Resolvent $S(c)$ is false in I and it is kept.

In these examples I is given by a finite set of literals: Example 1 is propositional, and in Example 3 the Herbrand base is finite, because there are no function symbols. The examples in [79] include a theorem from algebra, where the interpretation is given by a multiplication table and hence is really of semantic nature. The crux of semantic resolution is the *representation* of I. In theory, a Herbrand interpretation is given by a subset of the Herbrand base of S. In practice, one needs a *finite* representation of I, which is a non-trivial issue,

whenever the Herbrand base is not finite, or a mechanism to test the truth of a literal in I. Two instances of semantic resolution that aimed at addressing this issue are *hyperresolution* [74] and the *set-of-support strategy* [86].

Hyperresolution assumes that I contains either all negative literals or all positive literals. In the first case, it is called *positive* hyperresolution, because electrons and all resolvents are positive clauses: positive electrons are used to resolve away all negative literals in the nucleus to get a positive hyperresolvent. In the second case, it is called *negative* hyperresolution, because electrons and all resolvents are negative clauses: negative electrons are used to resolve away all positive literals in the nucleus to get a negative hyperresolvent. Example 1 is an instance of positive hyperresolution.

The set-of-support strategy assumes that $S = T \uplus SOS$, where SOS (for Set of Support) contains initially the clauses coming from the negation of the conjecture, and $T = S \setminus SOS$ is consistent, for some I such that $I \models T$ and $I \not\models SOS$. A resolution of two clauses is permitted, if at least one is from SOS, in order to avoid expanding the consistent set T. All resolvents are added to SOS. Thus, all inferences involve clauses descending from the negation of the conjecture: a method with this property is deemed *goal-sensitive*.

In terms of implementation, positive hyperresolution is often implemented in contemporary theorem provers by resolution with *selection of negative literals*. Indeed, resolution can be restricted by a *selection function* that selects negative literals [4]. A clause can have all, some, or none of its negative literals selected, depending on the selection function. In *resolution with negative selection*, the negative literal resolved upon must be selected, and the other parent must not contain selected literals. If some negative literal is selected for each clause containing one, one parent in each resolution inference will be a positive clause, that is, an electron for positive hyperresolution. Thus, 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.

The set-of-support strategy is available in all theorem provers that feature the *given-clause loop* [61], which is a de facto standard for resolution-based provers. This algorithm maintains two lists of clauses, named *to-be-selected* and *already-selected*, and at each iteration it extracts a *given clause* from *to-be-selected*. In its simplest version, with only resolution as inference rule, it performs all resolutions between the given clause and the clauses in *already-selected*; adds all resolvents to *to-be-selected*; and adds the given clause to *already-selected*. If one initializes these lists by putting the clauses in T in *already-selected*, and the clauses in SOS in *to-be-selected*, this algorithm implements the set-of-support strategy. Indeed, in the original version of the given-clause algorithm, *to-be-selected* was called *SOS*, and *already-selected* was called *Usable*.

State-of-the-art resolution-based theorem provers implement more sophisticated versions of the given clause algorithm, which also accomodate *contraction rules*, that delete (e.g., *subsumption*, *tautology deletion*) or simplify clauses (e.g., *clausal simplification*, *equational simplification*). The compatibility of contraction rules with semantic strategies is not obvious, as shown by the following:

Example 4. Let $T = \{\neg P,\ P \vee Q\}$ and $SOS = \{\neg Q\}$. Clausal simplification, which is a combination of resolution and subsumption, applies $\neg Q$ to simplify $P \vee Q$ to P. If the result is $T = \{\neg P,\ P\}$ and $SOS = \{\neg Q\}$, the consistent set T becomes inconsistent, and the refutational completeness of resolution with set-of-support collapses, since the set-of-support strategy does not allow us to resolve P and $\neg P$, being both in T. The correct application of clausal simplification yields $T = \{\neg P\}$ and $SOS = \{\neg Q,\ P\}$, so that the refutation can be found.

In other words, if a clause in SOS simplifies a clause, whether in T or in SOS, the resulting clause must be added to SOS. The integration of contraction rules and other enhancements, such as *lemmaizing*, in semantic strategies was investigated in general in [22].

Semantic resolution, hyperresolution, and the set-of-support strategy exhibit *semantic guidance*. We deem a method *semantically guided*, if it employs a *fixed* interpretation to drive the inferences. We deem a method *model-based*, if it builds and transforms a *candidate model*, and uses it to drive the inferences.

A beginning of the evolution from being semantically guided to being model-based can be traced back to the SCOTT system [80], which combined the finite model finder FINDER, that searches for small models, and the resolution-based theorem prover OTTER [61]. As the authors write "SCOTT brings semantic information gleaned from the proof attempt into the service of the syntax-based theorem prover." In SCOTT, FINDER provides OTTER with a *guide model*, which is used for an extended set-of-support strategy: in each resolution step at least one of the parent clauses must be false in the guide model. During the proof search FINDER updates periodically its model to make more clauses true. Thus, inferences are controlled as in the set-of-support strategy, but the guide model is *not fixed*, which is why SCOTT can be seen as a forerunner of model-based methods. Research on the cooperation between theorem prover and finite model finder continued with successors of OTTER, such as Prover9, and successors of FINDER, such as MACE4 [87]. This line of research has been especially fruitful in applications to mathematics (e.g., [3, 38]).

2.2 Hypertableaux

Tableau calculi offer an alternative to resolution and they have been discussed abundantly in the literature (e.g., Chapter 3 in [76]). Their advantages include no need for a clause normal form, a single proof object, and an easy extendability to other logics. The disadvantage, even in the case of clause normal form tableaux, is that variables are *rigid*, which means that substitutions have to be applied to all occurrences of a variable within the entire tableau. The *hypertableau calculus* [10] offers a more liberal treatment of variables, and borrows the concept of hyperinference from positive hyperresolution.

In this section, we adopt a Prolog-like notation for clauses: $A_1 \vee \ldots \vee A_m \vee \neg B_1 \vee \ldots \vee \neg B_n$ is written $A_1, \ldots, A_m \Leftarrow B_1, \ldots, B_n$, where $A_1, \ldots, A_m$ form the *head* of the clause and are called *head literals*, and $B_1, \ldots, B_n$ form the *body*. There are two rules for constructing a hypertableau (cf. [10]): the *initialization*

rule gives a tableau consisting of a single node labeled with $\top$; this one-element branch is *open*. The *hyperextension* rule selects an open branch and a clause $A_1, \ldots, A_m \Leftarrow B_1, \ldots, B_n$, where $m, n \geq 0$, from the given set S, such that there exists a most general unifier σ which makes all the $B_i\sigma$'s *follow logically* from the model given by the branch. If there is a variable in the clause that has an occurrence in more than one head literal A_i, a *purifying substitution* π is used to ground this variable. Then the branch is extended by new nodes labeled with $A_i\sigma\pi, \ldots, A_m\sigma\pi$. A branch is *closed* if it can be extended by a clause without head literals. S is unsatisfiable if and only if there is a hypertableau for S whose branches are all closed.

Two major advantages of hyperextension are that it avoids unnecessary branching, and only variables in the clauses are universally quantified and get instantiated, while variables in the branches are treated as *free* variables (except those occurring in different head literals). The latter feature allows a superposition-like handling of equality [11], while the former is relevant for hypertableaux for description logic [78], which we shall return to in the next section. Hypertableaux were implemented in the *Hyper* theorem prover for first-order logic, followed by *E-Hyper* implementing also the handling of equality.

Example 5. An example refutation is given in Figure 1. The initial tableau is set up with the only positive clause. Extension at $R(a)$ with the second clause uses $\sigma = \{x \leftarrow a\}$: since y appears only once in the resulting head, $\pi = \varepsilon$ and y remains as a free variable. In the right subtree $R(f(z))$ is extended with the second clause and $\sigma = \{x \leftarrow f(z)\}$. In the head $P(f(z)), Q(f(z), y)$ of the resulting clause z is repeated: an *instance generation* mechanism produces $\pi = \{z \leftarrow b\}$, or the instance $P(f(b)), Q(f(b), y) \Leftarrow R(f(b))$, to find a refutation. Note how the tableau contains by construction only positive literals, and the interpretation given by a branch is used to control the extension steps very much like in hyperresolution.

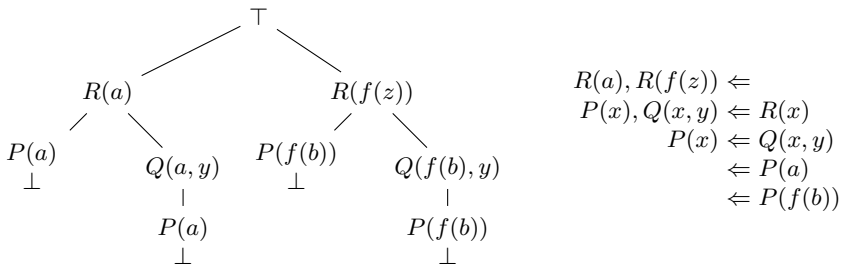

Fig. 1. A sample hypertableaux refutation with the clause set on the right.

2.3 Model-based Transformation of Clause Sets

Hypertableaux use partial models, that is, models for parts of a clause set, to control the search space. An open branch that cannot be expanded further represents a model for the entire clause set. In this section we present a transformation method, borrowed from model-based diagnosis and presented in [8], which is based on a given model and therefore can be installed on top of hypertableaux. In applications to diagnosis, one has a set of clauses S which corresponds to a description of a system, such as an electrical circuit. Very often there is a model I of a correctly functioning system available; in case of an electrical circuit it may be provided by the design tool itself. If the actual circuit is fed with an input and does not show the expected output, the task is to find a diagnosis, or those parts of the circuit which may be broken. Instead of doing reasoning with the system description S and its input and output in order to find the erroneous parts, the idea is to compute only *deviations* from the initially given model I.

Assume that S is a set of propositional clauses and I a set of propositional atoms; as a very simple example take

$$S = \{B \Leftarrow,\ C \Leftarrow A, B\} \text{ and } I = \{A\}.$$

Each clause in S is transformed by replacing a positive literal L by $\neg neg\_L$ and a negative literal $\neg L$ by neg_L, if L is contained in I. In other words, a literal which is contained in the initial model moves to the other side of the arrow and is renamed with the prefix $neg\_$ as in

$$S' = \{B \Leftarrow,\ C, neg\_A \Leftarrow B\}.$$

This transformation is model-preserving, as every model of S is a model of S'. For this it suffices to assign true to neg_L if and only if L is false, for every $L \in I$, and keep truth values unchanged for atoms outside of I. This property is independent of I, and it holds even if I is not a model of S. In our example, after initialization, first hyperextension with $B \Leftarrow$, and then hyperextension with $C, neg\_A \Leftarrow B$, yield the open branches $\{B, C\}$ and $\{B, neg\_A\}$. Hyperextension with $C, neg\_A \Leftarrow B$ can be applied because only B occurs in the body. Since A is assumed to be true in I, it can be added: adding A to $\{B, C\}$ yields model $\{A, B, C\}$; adding A to $\{B, neg\_A\}$ yields model $\{B\}$. If deriving A in S is very expensive, it pays off to save this derivation by moving A as neg_A to the body of the clause. In this example a Horn clause becomes non-Horn, introducing the case where A is false, and neg_A holds, although A is in I. Symmetrically, a non-Horn clause may become Horn. This transformation technique enabled a hypertableau prover to compute benchmarks from electrical engineering [8], and was also applied to the view update problem in databases [2].

Although this transformation mechanism only works in the propositional case, it can be extended to description logic [39]. Indeed, most description logic reasoners are based on tableau calculi, and a hypertableau calculus was used in [78] as a basis for an efficient reasoner for the description logic $\mathcal{SHIQ}$. For this purpose, the authors define *DL-clauses* as clauses without occurrences of

function symbols, and such that the head is allowed to include disjunctions of atoms, which may contain existential role restrictions as in

$$\exists repairs.Car(x) \Leftarrow Mechanic(x).$$

In other words, a given $\mathcal{SHIQ}$-Tbox is translated to a large extent into first-order logic; only existential role restrictions are kept as positive "literals." Given a Tbox in the form of a set of DL-clauses, if we have in addition an Abox, or a set of ground assertions, we can use the interpretation given by the ABox as initial model for the model-based transformation [39]. On this basis, the already mentioned *E-Hyper* reasoner was modified to become *E-KRHyper*, which was shown to be a decision procedure for $\mathcal{SHIQ}$ in [16].

2.4 The Model Evolution Calculus

The practical success of DPLL-based SAT solvers suggested the goal of lifting features of DPLL to the first-order level. Research focused on *splitting first-order clauses*, seen as a way to improve the capability to handle non-Horn clauses. Breaking first-order clauses apart is not as simple as in propositional logic, because a clause stands for all its ground instances, and literals share variables that are implicitly universally quantified. Decomposing disjunction is a native feature in tableaux, whose downside is represented by rigid variables, as already discussed in Section 2.2, where we saw how hypertableaux offer a possible answer.

The quest for ways to split efficiently clauses such as $A(x) \vee B(x)$ led to the *model evolution calculus* [13]. In this method splitting $A(x) \vee B(x)$ yields a branch with $A(x)$, meaning $\forall x A(x)$, and one with $\neg A(c)$, the Skolemized form of $\neg\forall x A(x) \equiv \exists x \neg A(x)$. Splitting in this way has the disadvantage that the signature changes, and Skolem constants, being new, do not unify with other non-variable terms. Thus, the model evolution calculus employs *parameters*, in place of Skolem constants, to replace existentially quantified variables. These parameters are similar to the free variables of hypertableaux.

The similarity between the model evolution calculus and DPLL goes beyond splitting, as the model evolution calculus aims at being a faithful lifting of DPLL to first-order logic. Indeed, a central feature of the model evolution calculus is that it maintains a *context* Λ, which is a finite set of literals, representing a Herbrand interpretation I_Λ, seen as a candidate partial model of the input set of clauses S. Thus, the model evolution calculus is a *model-based* first-order method. Literals in Λ may contain variables, implicitly universally quantified as in clauses, and parameters. Clauses are written in the form $\Lambda \vdash C$, so that each clause carries the context with itself.

In order to determine whether $I_\Lambda \models L$, for L an atom in the Herbrand base of S, one looks at the most specific literal in Λ that subsumes L; in case of a tie, L is picked with positive sign. If I_Λ is not a model of S, the inference system unifies input clauses against Λ to find instances that are not true in I_Λ: these instances are subject to splitting, to modify Λ and repair I_Λ. Otherwise, the system recognizes that Λ cannot be fixed and declares S unsatisfiable. As

DPLL uses *depth-first search with backtracking*, the model evolution calculus uses depth-first search with backtracking and *iterative deepening* on term depth, which however may skew the search towards big proofs with small term depth. The model evolution calculus was implemented in the *Darwin* prover [9], and extended to handle equality on its own [12] and with superposition [14].

2.5 SGGS: Semantically-Guided Goal-Sensitive Reasoning

SGGS, for *Semantically-Guided Goal-Sensitive* reasoning, is a new theorem-proving method for first-order logic [29, 26, 28, 27], which inherits features from several of the strategies that we surveyed in the previous sections. SGGS is *semantically guided* by a fixed initial interpretation I like semantic resolution; and it is *goal-sensitive* like the set-of-support strategy. With hyperresolution and hypertableaux, it shares the concept of hyperinference, although the hyperinference in SGGS, as we shall see, is an instance generation inference, and therefore its closest ancestor is *hyperlinking* [58, 71], an inference rule that uses the most general unifier of a hyperresolution step to generate instances of the parents, rather than a hyperresolvent.

Most importantly, SGGS is *model-based* at the first-order level, in the sense of working by representing and transforming a candidate partial model of the given set S of first-order clauses. This fundamental characteristic is in common with the model evolution calculus, but while the latter lifts DPLL, SGGS lifts DPLL-CDCL to first-order logic, and it combines the model-based character with the semantic guidance and the goal sensitivity. Indeed, SGGS was motivated by the quest for a method that is simultaneously first-order, model-based, semantically-guided, and goal-sensitive. Furthermore, SGGS is *proof confluent*, which means it does not need backtracking, and it does not necessarily reduce to either DPLL or DPLL-CDCL, if given a propositional problem.

In DPLL-CDCL, if a literal L appears in the trail that represents the candidate partial model, all occurrences of $\neg L$ in the set of clauses are false. If all literals of a clause C are false, C is in *conflict*; if all literals of C except one, say Q, are false, Q is an *implied literal* with C as *justification*. The status of C *depends* on the *decision levels* where the complements of its literals were either guessed (decision) or implied (Boolean propagation). SGGS generalizes these concepts to first-order logic. Since variables in first-order literals are implicitly universally quantified, if L is true, $\neg L$ is false, but if L is false, we only know that a ground instance of $\neg L$ is true. SGGS restores the symmetry by introducing the notion of *uniform falsity*: L is uniformly false, if all its ground instances are false, or, equivalently, if $\neg L$ is true. A first rôle of the given interpretation I is to provide a *reference model* where to evaluate the truth value of literals: a literal is *I-true*, if it is true in I, and *I-false*, if it is uniformly false in I.

An *SGGS clause sequence* Γ is a sequence of clauses, where every literal is either I-true or I-false, so that it tells the truth value in I of all its ground instances. In every clause C in Γ a literal is *selected*: if $C = L_1 \vee \ldots \vee L_n$ and L_n is selected, we write the clause as $L_1 \vee \ldots \vee [L_n]$, or, more compactly, $C[L_n]$, with a slight abuse of the notation. SGGS tries to modify I into a model of S

(if I is a model of S the problem is solved). Thus, I-false literals are preferred for selection, and an I-true literal is selected only in a clause whose literals are all I-true, called *I-all-true* clause. A second rôle of the given interpretation I is to provide a *starting point* for the search of a model for S.

An SGGS clause sequence Γ represents a *partial interpretation* $I^p(\Gamma)$: if Γ is the empty sequence, denoted by ε, $I^p(\Gamma)$ is empty; if Γ is $C_1[L_1], \ldots, C_i[L_i]$, and $I^p(\Gamma|_{i-1})$ is the partial interpretation represented by $C_1[L_1], \ldots, C_{i-1}[L_{i-1}]$, then $I^p(\Gamma)$ is $I^p(\Gamma|_{i-1})$ plus the ground instances $L_i\sigma$ of L_i, such that $C_i\sigma$ is ground, $C_i\sigma$ is not satisfied by $I^p(\Gamma|_{i-1})$, and $\neg L_i\sigma$ is not in $I^p(\Gamma|_{i-1})$, so that $L_i\sigma$ can be added to satisfy $C_i\sigma$. In other words, each clause adds the ground instances of its selected literal that satisfy ground instances of the clause not satisfied thus far.

An *interpretation* $I[\Gamma]$ is obtained by consulting first $I^p(\Gamma)$, and then I: for a ground literal L, if its atom appears in $I^p(\Gamma)$, its truth value in $I[\Gamma]$ is that in $I^p(\Gamma)$; otherwise, it is that in I. Thus, $I[\Gamma]$ is I modified to satisfy the clauses in Γ by satisfying the selected literals, and since I-true selected literals are already true in I, the I-false selected literals are those that matter. For example, if Γ is $[P(x)]$, $\neg P(f(y)) \vee [Q(y)]$, $\neg P(f(z)) \vee \neg Q(g(z)) \vee [R(f(z), g(z))]$, and I is all negative like in positive hyperresolution, $I[\Gamma]$ satisfies all ground instances of $P(x)$, $Q(y)$, and $R(f(z), g(z))$, and no other positive literal.

SGGS generalizes Boolean, or clausal, propagation to first-order logic. Consider an I-false (I-true) literal M selected in clause C_j in Γ, and an I-true (I-false) literal L in C_i, $i > j$: if all ground instances of L appear negated among the ground instances of M added to $I^p(\Gamma)$, L is uniformly false in $I[\Gamma]$ because of M, and *depends* on M, like $\neg L$ *depends* on L in propositional Boolean propagation, when L is in the trail. If this happens for *all* its literals, clause $C[L]$ is in *conflict* with $I[\Gamma]$; if this happens for all its literals except L, L is an *implied literal* with $C[L]$ as *justification*. SGGS employs *assignment functions* to keep track of the *dependencies* of I-true literals on selected I-false literals, realizing a sort of *first-order propagation modulo semantic guidance* by I. SGGS ensures that I-all-true clauses in Γ are either conflict clauses or justifications.

The main inference rule of SGGS, called *SGGS-extension*, uses the current clause sequence Γ and a clause C in S to generate an instance E of C and add it to Γ to obtain the next clause sequence Γ'. SGGS-extension is a hyperinference, because it unifies literals $L_1, \ldots, L_n$ of C with I-false selected literals $M_1, \ldots, M_n$ of opposite sign in Γ. The hyperinference is *guided* by $I[\Gamma]$, because I-false selected literals contribute to $I[\Gamma]$ as explained above. Another ingredient of the instance generation mechanism ensures that every literal in E is either I-true or I-false. SGGS-extension is also responsible for selecting a literal in E.

The *lifting theorem* for SGGS-extension shows that if $I[\Gamma] \not\models C'$ for some ground instance C' of a clause $C \in S$, SGGS-extension builds an instance E of C such that C' is an instance of E. There are three kinds of SGGS-extension: (1) add a clause E which is in conflict with $I[\Gamma]$ and is I-all-true; (2) add a clause E which is in conflict with $I[\Gamma]$ but is not I-all-true; and (3) add a clause E which is not in conflict with $I[\Gamma]$. In cases (1) and (2), it is necessary to *solve*

the conflict: it is here that SGGS lifts the conflict-driven clause learning (CDCL) mechanism of DPLL-CDCL to the first-order level.

In DPLL-CDCL a conflict is *explained* by resolving a conflict clause C with the justification D of a literal whose complement is in C, generating a new conflict clause. Typically resolution continues until we get either the empty clause $\bot$ or an *asserting clause*, namely a clause where only one literal Q is falsified in the current decision level. DPLL-CDCL learns the asserting clause and backjumps to the shallowest level where Q is undefined and all other literals in the asserting clause are false, so that Q enters the trail with the asserting clause as justification. SGGS *explains* a conflict by resolving the conflict clause E with an I-all-true clause $D[M]$ in Γ which is the justification of the literal M that makes an I-false literal L in E uniformly false in $I[\Gamma]$. Resolution continues until we get either $\bot$ or a conflict clause $E[L]$ which is I-all-true. If $\bot$ arises, S is unsatisfiable. Otherwise, SGGS *moves* the I-all-true clause $E[L]$ to the left of the clause $B[M]$, whose I-false selected literal M makes L uniformly false in $I[\Gamma]$. The effect is to *flip* at once the truth value of *all* ground instances of L in $I[\Gamma]$, so that the conflict is solved, L is implied, and $E[L]$ satisfied.

In order to simplify the presentation, up to here we omitted that clauses in SGGS may have *constraints*. For example, $x \not\equiv y \rhd P(x,y) \vee Q(y,x)$ is a *constrained clause*, which represents its ground instances that satisfy the constraints: $P(a,b) \vee Q(b,a)$ is an instance, while $P(a,a) \vee Q(a,a)$ is not. The reason for constraints is that selected literals of clauses in Γ may *intersect*, in the sense of having ground instances with the same atoms. Since selected literals determine $I^p(\Gamma)$, whence $I[\Gamma]$, non-empty intersections represent *duplications*, if the literals have the same sign, and *contradictions*, otherwise. SGGS removes duplications by deletion of clauses, and contradictions by resolution. However, before doing either, it needs to *isolate* the shared ground instances in the selected literal of *one* clause. For this purpose, SGGS features inference rules that replace a clause by a *partition*, that is, a set of clauses that represent the same ground instances and have *disjoint* selected literals. This requires constraints. For example, a partition of $[P(x,y)] \vee Q(x,y)$ is $\{true \rhd [P(f(z),y)] \vee Q(f(z),y),\ top(x) \neq f \rhd [P(x,y)] \vee Q(x,y)\}$, where the constraint $top(x) \neq f$ means that variable x cannot be instantiated with a term whose topmost symbol is f. If L and M in $C[L]$ and $D[M]$ of Γ intersect, SGGS partitions $C[L]$ by $D[M]$: it partitions $C[L]$ into $A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$ so that only L_j, for some j, $1 \leq j \leq n$, intersects with M, and $A_j \rhd C_j[L_j]$ is either deleted or resolved with $D[M]$.

The following example shows an SGGS-refutation:

Example 6. Given $S = \{\neg P(f(x)) \vee \neg Q(g(x)) \vee R(x),\ P(x),\ Q(y),\ \neg R(c)\}$, let I be all negative. An SGGS-derivation starts with the empty sequence. Then, four SGGS-extension steps apply:

$$
\begin{array}{ll}
\Gamma_0\colon & \varepsilon \\
\Gamma_1\colon & [P(x)] \\
\Gamma_2\colon & [P(x)],\ [Q(y)] \\
\Gamma_3\colon & [P(x)],\ [Q(y)],\ \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)] \\
\Gamma_4\colon & [P(x)],\ [Q(y)],\ \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]
\end{array}
$$

At this stage, the selected literals $R(x)$ and $\neg R(c)$ intersect, and therefore SGGS partitions $\neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)]$ by $[\neg R(c)]$:

$$\begin{aligned}\Gamma_5\colon\ & [P(x)],\ [Q(y)],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\\ & \neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)],\ [\neg R(c)]\end{aligned}$$

Now the I-all-true clause $\neg R(c)$ is in conflict with $I[\Gamma_5]$. Thus, SGGS moves it left of the clause $\neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)]$ that makes $\neg R(c)$ false in $I[\Gamma_5]$, in order to amend the induced interpretation. Then, it resolves these two clauses, and replaces the parent that is not I-all-true, namely $\neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)]$, by the resolvent $\neg P(f(c)) \vee \neg Q(g(c))$:

$$\begin{aligned}\Gamma_6\colon\ & [P(x)],\ [Q(y)],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)],\\ & \neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)]\\ \Gamma_7\colon\ & [P(x)],\ [Q(y)],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)],\\ & \neg P(f(c)) \vee [\neg Q(g(c))]\end{aligned}$$

Assuming that in the resolvent the literal $\neg Q(g(c))$ gets selected, there is now an intersection between selected literals $\neg Q(g(c))$ and $Q(y)$, so that SGGS partitions $Q(y)$ by $\neg P(f(c)) \vee \neg Q(g(c))$:

$$\begin{aligned}\Gamma_8\colon\ & [P(x)],\ top(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],\ [Q(g(c))],\\ & x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)],\ \neg P(f(c)) \vee [\neg Q(g(c))]\end{aligned}$$

At this point, the I-all-true clause $\neg P(f(c)) \vee [\neg Q(g(c))]$ is in conflict with $I[\Gamma_8]$. As before, SGGS moves it left of the clause that makes its selected literal $\neg Q(g(c))$ false, namely $[Q(g(c))]$, in order to fix the candidate model, and then resolves $\neg P(f(c)) \vee [\neg Q(g(c))]$ and $[Q(g(c))]$, replacing the latter by the resolvent $\neg P(f(c))$:

$$\begin{aligned}\Gamma_9\colon\ & [P(x)],\ top(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],\ \neg P(f(c)) \vee [\neg Q(g(c))],\\ & [Q(g(c))],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]\\ \Gamma_{10}\colon\ & [P(x)],\ top(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],\ \neg P(f(c)) \vee [\neg Q(g(c))],\\ & [\neg P(f(c))],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]\end{aligned}$$

The resolvent has only one literal which gets selected; since $[\neg P(f(c))]$ intersects with $[P(x)]$, the next inference partitions $[P(x)]$ by $[\neg P(f(c))]$:

$$\begin{aligned}\Gamma_{11}\colon\ & top(x) \neq f \rhd [P(x)],\ y \not\equiv c \rhd [P(f(y))],\ [P(f(c))],\ top(y) \neq g \rhd [Q(y)],\\ & z \not\equiv c \rhd [Q(g(z))],\ \neg P(f(c)) \vee [\neg Q(g(c))],\ [\neg P(f(c))],\\ & x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]\end{aligned}$$

The next step moves the I-all-true clause $[\neg P(f(c))]$, which is in conflict with $I[\Gamma_{11}]$, to the left of the clause $[P(f(c))]$ that makes $[\neg P(f(c))]$ false in $I[\Gamma_{11}]$, and then resolves these two clauses to generate the empty clause:

$$\begin{aligned}\Gamma_{12}\colon\ & top(x) \neq f \rhd [P(x)],\ y \not\equiv c \rhd [P(f(y))],\ [\neg P(f(c))],\ [P(f(c))],\\ & top(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],\ \neg P(f(c)) \vee [\neg Q(g(c))],\\ & x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]\\ \Gamma_{13}\colon\ & top(x) \neq f \rhd [P(x)],\ y \not\equiv c \rhd [P(f(y))],\ [\neg P(f(c))],\ \bot,\\ & top(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],\ \neg P(f(c)) \vee [\neg Q(g(c))],\\ & x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]\end{aligned}$$

This example only illustrates the basic mechanisms of SGGS. This method is so new that it has not yet been implemented: the hope is that its conflict-driven model-repair mechanism will have on first-order theorem proving an effect similar to that of the transition from DPLL to DPLL-CDCL for SAT-solvers. If this were true, even in part, the benefit could be momentous, considering that CDCL played a key rôle in the success of SAT technology. Another expectation is that non-trivial semantic guidance (i.e., not based on sign like in hyperesolution) pays off in case of many axioms or large knowledge bases.

3 Model-based Reasoning in First-Order Theories

There are basically two ways one can think about a theory presented by a set of axioms: as the set of all theorems that are logical consequences of the axioms, or as the set of all interpretations that are models of the axioms. The two are obviously connected, but may lead to different styles of reasoning, that we portray by the selection of methods in this section. We cover approaches that *build axioms* into resolution, *hierarchical* and *locality-based* theory reasoning, and a recent method called *Model-Constructing satisfiability calculus* or *MCsat*.

3.1 Building Theory Axioms into Resolution and Superposition

The early approaches to theory reasoning emphasized the axioms, by building them into the inference systems. The first analyzed theory was *equality*: since submitting the equality axioms to resolution, or other inference systems for first-order logic, leads to an explosion of the search space, *paramodulation*, *superposition*, and *rewriting* were developed to build equality into resolution (e.g., [73, 48, 77, 4, 21] and Chapters 7 and 9 in [76]).

Once equality was conquered, research flourished on building-in theories (e.g., [72, 66, 54, 49, 36, 40, 53, 30]). *Equational theories*, that are axiomatized by sets of equalities, and among them *permutative theories*, where the two sides of each axiom are permutations of the same symbols, as in *associativity* and *commutativity*, received the most attention. A main ingredient is to replace syntactic unification by unification *modulo* a set E of equational axioms, a concept generalized by José Meseguer to *order-sorted E-unification* (e.g., [43, 37, 46]). This kind of approach was pursued further, by building into superposition axioms for *monoids* [42], *groups* [85], *rings* and *modules* [84], or by generalizing superposition to embed *transitive relations* other than equality [5]. The complexities and limitations of these techniques led to investigate the methods for *hierarchical* theory reasoning that follow.

3.2 Hierarchical Reasoning by Superposition

Since José Meseguer's work with Joe Goguen (e.g., [44]), it became clear that a major issue at the cross-roads of reasoning, specifying, and programming, is that theories, or specifications, are built by *extension* to form *hierarchies*. A

base theory $\mathcal{T}_0$ is defined by a set of *sorts* $\mathcal{S}_0$, a *signature* Σ_0, possibly a set of axioms N_0, and the class $\mathcal{C}_0$ of its *models* (e.g., term-generated Σ_0-algebras). An *extended* or *enriched* theory $\mathcal{T}$ adds new sorts ($\mathcal{S}_0 \subseteq \mathcal{S}$), new function symbols ($\Sigma_0 \subseteq \Sigma$), called *extension functions*, and new axioms ($N_0 \subseteq N$), specifying properties of the new symbols. For the base theory the class of models is given, while the extension is defined axiomatically. A pair $(\mathcal{T}_0, \mathcal{T})$ as above forms a *hierarchy* with *enrichment axioms* N.

The crux of extending specifications was popularized by Joe Goguen and José Meseguer as *no junk and no confusion*: an interpretation of $\mathcal{S}$ and Σ, which is a model of N, is a model of $\mathcal{T}$ only if it extends a model in $\mathcal{C}_0$, without collapsing its sorts, or making distinct elements equal (*no confusion*), or introducing new elements of base sort (*no junk*). A sufficient condition for the latter is *sufficient completeness*, a property studied also in inductive theorem proving, which basically says that every ground non-base term t' of base sort is equal to a ground base term t. Sufficient completeness is a strong restriction, violated by merely adding a constant symbol: if $\Sigma_0 = \{a, b\}$, $N = N_0 = \{a \not\simeq b\}$, and $\Sigma = \{a, b, c\}$, where a, b, and c are constants of the same sort, the extension is not sufficiently complete, because c is junk, or a model with three distinct elements is not isomorphic to one with two. Although sufficient completeness is undecidable in general (e.g., [57]), sufficient completeness analyzers exist (e.g., [56, 45, 47]), with key contributions by José Meseguer.

Hierarchic superposition was introduced in [6] and developed in [41] to reason about a hierarchy $(\mathcal{T}_0, \mathcal{T})$ with enrichment axioms N, where N is a set of clauses. We assume to have a decision procedure to detect that a finite set of Σ_0-clauses is $\mathcal{T}_0$-unsatisfiable. Given a set S of Σ-clauses, the problem is to determine whether S is false in all models of the hierarchic specification, or, equivalently, whether $N \cup S$ has no model whose reduct to Σ_0 is a model of $\mathcal{T}_0$. The problem is solved by using the $\mathcal{T}_0$-reasoner as a black-box to take care of the base part, while superposition-based inferences apply only to non-base literals.[3] First, for every clause C, whenever a subterm t whose top symbol is a base operator occurs immediately below a non-base operator symbol (or vice versa), t is replaced by a new variable x and the equation $x \simeq t$ is added to the antecedent of C. This transformation is called *abstraction*. Then, the inference rules are modified to require that all substitutions are *simple*, meaning that they map variables of base sort to base terms. A meta-rule named *constraint refutation* detects that a finite set of Σ_0-clauses is inconsistent in $\mathcal{T}_0$ by invoking the $\mathcal{T}_0$-reasoner. Hierarchic superposition was proved refutationally complete in [6], provided $\mathcal{T}_0$ is compact, which is a basic preliminary to make constraint refutation mechanizable, and $N \cup S$ is *sufficiently complete with respect to simple instances*, which means that for every model I of all simple ground instances of the clauses in $N \cup S$, and every ground non-base term t', there exists a ground base term t (which may depend on I) such that $I \models t' \simeq t$.

[3] Other approaches to subdivide work between superposition and an SMT-solver appeared in [20, 25].

There are situations where the enrichment adds *partial* functions: Σ_0 contains only total function symbols, while $\Sigma \setminus \Sigma_0$ may contain partial functions and total functions having as codomain a new sort. Hierarchic superposition was generalized to handle both total and partial function symbols, yielding a *partial hierarchic superposition calculus* [41]. To have an idea of the difficulties posed by partial functions, consider that replacement of equals by equals may be unsound in their presence. For example, $s \not\simeq s$ may hold in a partial algebra (i.e., a structure where some function symbols are interpreted as partial), if s is undefined. Thus, the equality resolution rule (e.g., resolution between $C \vee s \not\simeq s$ and $x \simeq x$) is restricted to apply only if s is guaranteed to be defined. Other restrictions impose that terms replaced by inferences may contain a partial function symbol only at the top; substitutions cannot introduce partial function symbols; and every ground term made only of total symbols is smaller than any ground term containing a partial function symbol in the ordering used by the inference system. The following example portrays the partial function case:

Example 7. Let $\mathcal{T}_0$ be the base theory defined by $\mathcal{S}_0 = \{\mathsf{data}\}$, $\Sigma_0 = \{b\colon \to \mathsf{data}, f\colon \mathsf{data} \to \mathsf{data}\}$, and $N_0 = \{\forall x\, f(f(x)) \simeq f(x)\}$. We consider the extension with a new sort list, total functions $\{\mathsf{cons}\colon \mathsf{data}, \mathsf{list} \to \mathsf{list}, \mathsf{nil}\colon \to \mathsf{list}, d :\to \mathsf{list}\}$, partial functions $\{\mathsf{car} : \mathsf{list} \to \mathsf{data}, \mathsf{cdr} : \mathsf{list} \to \mathsf{list}\}$, and the following clauses, where $N = \{(1), (2), (3)\}$ and $S = \{(4), (5)\}$:

$$
\begin{array}{ll}
(1) & \mathsf{car}(\mathsf{cons}(x, l)) \simeq x \\
(2) & \mathsf{cdr}(\mathsf{cons}(x, l)) \simeq l \\
(3) & \mathsf{cons}(\mathsf{car}(l), \mathsf{cdr}(l)) \simeq l \\
(4) & f(b) \simeq b \\
(5) & f(f(b)) \not\simeq \mathsf{car}(\mathsf{cdr}(\mathsf{cons}(f(b), \mathsf{cons}(b, d))))
\end{array}
$$

The partial hierarchic superposition calculus deduces:

$$
\begin{array}{lll}
(6)\ x \not\simeq f(f(b)) \vee y \not\simeq f(b) \vee z \not\simeq b \vee x \not\simeq \mathsf{car}(\mathsf{cdr}(\mathsf{cons}(y, \mathsf{cons}(z, d)))) & \text{Abstr. } (5) \\
(7)\ x \not\simeq f(f(b)) \vee y \not\simeq f(b) \vee z \not\simeq b \vee x \not\simeq \mathsf{car}(\mathsf{cons}(z, d)) & \text{Superp. } (2),(6) \\
(8)\ x \not\simeq f(f(b)) \vee y \not\simeq f(b) \vee z \not\simeq b \vee x \not\simeq z & \text{Superp. } (1),(7) \\
(9)\ \bot & \text{Constraint refutation } (4),(8)
\end{array}
$$

Under the assumption that $\mathcal{T}_0$ is a universal first-order theory, which ensures compactness, the partial hierarchic superposition calculus was proved sound and complete in [41]: if a contradiction cannot be derived from $N \cup S$ using this calculus, then $N \cup S$ has a model which is a partial algebra. Thus, if the unsatisfiability of $N \cup S$ does not depend on the totality of the extension functions, the partial hierarchic superposition calculus can detect its inconsistency. In certain problem classes where partial algebras can always be made total, the calculus is complete also for total functions. Research on hierarchic superposition continued in [1], where an implementation for extensions of linear arithmetic was presented, and in [15], where the calculus was made "more complete" in practice.

3.3 Hierarchical Reasoning in Local Theory Extensions

A series of papers starting with [81] identified a class of theory extensions $(\mathcal{T}_0, \mathcal{T})$, called *local*, which admit a complete hierarchical method for checking satisfiabil-

ity of *ground* clauses, without requiring either sufficient completeness or that $\mathcal{T}_0$ is a universal first-order theory. The enrichment axioms in N do not have to be clauses: if they are, we have an extension *with clauses*; if N consists of formulæ of the form $\forall \bar{x}\,(\Phi(\bar{x}) \vee D(\bar{x}))$, where $\Phi(\bar{x})$ is an *arbitrary* Σ_0-formula and $D(\bar{x})$ is a Σ-clause, with at least one occurrence of an extension function, we have an extension *with augmented clauses*. The basic assumption that $\mathcal{T}_0$, or a fragment thereof, admits a decision procedure for satisfiability clearly remains.

As we saw throughout this survey, instantiating universally quantified variables is crucial in first-order reasoning. Informally, a theory extension is *local*, if it is sufficient to consider only a *finite* set of instances. Let G be a set of ground clauses to be refuted in $\mathcal{T}$, and let $N[G]$ denote the set of instances of the clauses in N where every term whose top symbol is an extension function is a ground term occurring in N or G. Theory $\mathcal{T}$ is a *local extension* of $\mathcal{T}_0$, if $N[G]$ suffices to prove the $\mathcal{T}$-unsatisfiability of G [81]. Subsequent papers studied variants of locality, including those for extensions with augmented clauses, and for combinations of local theories, and proved that locality can be recognized by showing that certain partial algebras embed into total ones [81, 82, 50, 51].

If $\mathcal{T}$ is a local extension, it is possible to check the $\mathcal{T}$-satisfiability of G by hierarchical reasoning [81, 82, 50, 51], allowing the introduction of new constants by *abstraction* as in [64]. By locality, G is $\mathcal{T}$-unsatisfiable if and only if there is no model of $N[G] \cup G$ whose restriction to Σ_0 is a model of $\mathcal{T}_0$. By abstracting away non-base terms, $N[G] \cup G$ is transformed into an equisatisfiable set $N_0 \cup G_0 \cup D$, where N_0 and G_0 are sets of Σ_0-clauses, and D contains the definitions introduced by abstraction, namely equalities of the form $f(g_1, \ldots, g_n) \simeq c$, where f is an extension function, $g_1, \ldots, g_n$ are ground terms, and c is a new constant. The problem is reduced to that of testing the $\mathcal{T}_0$-satisfiability of $N_0 \cup G_0 \cup \mathsf{Con}_0$, where Con_0 contains the instances of the congruence axioms for the terms in D:

$$\mathsf{Con}_0 = \{\bigwedge_{i=1}^{n} c_i \simeq d_i \Rightarrow c \simeq d \mid f(c_1, \ldots, c_n) \simeq c, f(d_1, \ldots, d_n) \simeq d \in D\},$$

which can be solved by a decision procedure for $\mathcal{T}_0$ or a fragment thereof.

In the following example $\mathcal{T}_0$ is the theory of *linear arithmetic* over the real numbers, and $\mathcal{T}$ is its extension with a monotone unary function f, which is known to be a local extension [81]:

Example 8. Let G be $(a \leq b \wedge f(a) = f(b) + 1)$. The enrichment $N = \{x \leq y \Rightarrow f(x) \leq f(y)\}$ consists of the monotonicity axiom. In order to check whether G is $\mathcal{T}$-satisfiable, we compute $N[G]$, omitting the redundant clauses $c \leq c \Rightarrow f(c) \leq f(c)$ for $c \in \{a, b\}$:

$$N[G] = \{a \leq b \Rightarrow f(a) \leq f(b),\ b \leq a \Rightarrow f(b) \leq f(a)\}.$$

The application of abstraction to $N[G] \cup G$ yields $N_0 \cup G_0 \cup D$, where:

$$N_0 = \{a \leq b \Rightarrow a_1 \leq b_1,\ b \leq a \Rightarrow b_1 \leq a_1\}, \qquad G_0 = \{a \leq b,\ a_1 \simeq b_1 + 1\},$$

$D = \{a_1 \simeq f(a),\ b_1 \simeq f(b)\}$, and a_1 and b_1 are new constants. Thus, Con_0 is $\{a \simeq b \Rightarrow a_1 \simeq b_1\}$. A decision procedure for linear arithmetic applied to $N_0 \cup G_0 \cup \mathsf{Con}_0$ detects unsatisfiability.

3.4 Beyond SMT: Satisfiability Modulo Assignment and MCsat

Like SGGS generalizes conflict-driven clause learning (CDCL) to first-order logic and Herbrand interpretations, the *Model-Constructing satisfiability calculus*, or *MCsat* for short, generalizes CDCL to decidable fragments of first-order theories and their models [35, 55].

Recall that in DPLL-CDCL the trail that represents the candidate partial model contains only propositional literals; the inference mechanism that explains conflicts is propositional resolution; and learnt clauses are made of input atoms. These three characteristics are true also of the DPLL($\mathcal{T}$) paradigm for SMT-solvers [7], where an *abstraction function* maps finitely many input first-order ground atoms to finitely many propositional atoms. In this way, the method bridges the gap between the first-order language of the theory $\mathcal{T}$ and the propositional language of the DPLL-CDCL core solver. In DPLL($\mathcal{T}$), also $\mathcal{T}$-lemmas are made of input atoms, and the guarantee that no new atoms are generated is a key ingredient of the proof of termination of the method in [65].

Also when $\mathcal{T}$ is a union of theories $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$, the language of atoms remains finite. The standard method to combine satisfiability procedures for theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$ to get a satisfiability procedure for their union is *equality sharing* [64], better known as *Nelson-Oppen scheme*, even if equality sharing was the original name given by Greg Nelson, as reconstructed in [63]. Indeed, a key feature of equality sharing is that the combined procedures only need to share equalities between constant symbols. These equalities are mapped by the abstraction function to *proxy variables*, that is, propositional variables that stand for the equalities. As there are finitely many constant symbols, there are also finitely many proxy variables.

MCsat generalizes both model representation and inference mechanism beyond satisfiability modulo theories (SMT), because it is designed to decide a more general problem called *satisfiability modulo assignment* (SMA). An SMA problem consists of determining the satisfiability of a formula S in a theory $\mathcal{T}$, given an initial assignment I to some of the variables occuring in S, including *both* propositional variables and *free first-order variables*. SMT can be seen as a special case of SMA where I is empty. Also, since an SMT-solver builds partial assignments during the search for a satisfying one, an intermediate state of an SMT search can be viewed as an instance of SMA. A first major generalization of MCsat with respect to DPLL-CDCL and DPLL($\mathcal{T}$) is to allow the trail to contain also *assignments to free first-order variables* (e.g., $x \leftarrow 3$). Such assignments can be *semantic decisions* or *semantic propagations*, thus called to distinguish them from the Boolean decisions and Boolean propagations that yield the standard Boolean assignments (e.g., $L \leftarrow \mathit{true}$).

The answer to an SMA problem is either a model of S including the initial assignment I, or "unsatisfiable" with an *explanation*, that is, a formula

S' that follows from S and is inconsistent with I. This notion of explanation is a generalization of the explanation of conflicts by propositional resolution in DPLL-CDCL. Indeed, a second major generalization of MCsat with respect to DPLL-CDCL and DPLL($\mathcal{T}$) is to allow the inference mechanism that explains conflicts to generate *new atoms*, as shown in the following example in the quantifier-free fragment of the theory of equality:

Example 9. Assume that S is a conjunction of literals including $\{v \simeq f(a),\ w \simeq f(b)\}$, where a and b are constant symbols, f is a function symbol, and v and w are free variables. If the trail contains the assignments $a \leftarrow \alpha,\ b \leftarrow \alpha,\ w \leftarrow \beta_1,\ v \leftarrow \beta_2$, where α, β_1, and β_2 denote distinct values of the appropriate sorts, there is a conflict. The explanation is the formula $a \simeq b \Rightarrow f(a) \simeq f(b)$, which is an instance of the substitutivity axiom, or congruence axiom, for function f. Note how the atoms $a \simeq b$ and $f(a) \simeq f(b)$ need not appear in S, and therefore such a lemma could not be generated in DPLL($\mathcal{T}$).

In order to apply MCsat to a theory $\mathcal{T}$, one needs to give clausal inference rules to *explain* conflicts in $\mathcal{T}$. These inference rules generate clauses that may contain *new* (i.e., non-input) ground atoms in the signature of the theory. New atoms come from a *basis*, defined as the closure of the set of input atoms with respect to the inference rules. The proof of termination of the MCsat transition rules in [35] requires that the basis be *finite*. The following example illustrates the importance of this finiteness requirement:

Example 10. Given $S = \{x \geq 2,\ \neg(x \geq 1) \vee y \geq 1,\ x^2 + y^2 \leq 1 \vee xy > 1\}$, and starting with an empty trail $M = \emptyset$, a Boolean propagation puts $x \geq 2$ in the trail. Theory propagation adds $x \geq 1$, because $x \geq 2$ implies $x \geq 1$ in the theory, and $x \geq 1$ appears in S. A Boolean propagation over clause $\neg(x \geq 1) \vee y \geq 1$ adds $y \geq 1$, so that we have $M = x \geq 2,\ x \geq 1,\ y \geq 1$. If a Boolean decision guesses next $x^2 + y^2 \leq 1$ and then a semantic decision adds $x \leftarrow 2$, we have $M = x \geq 2,\ x \geq 1,\ y \geq 1,\ x^2 + y^2 \leq 1,\ x \leftarrow 2$ and a conflict, as there is no value for y such that $4 + y^2 \leq 1$. Learning $\neg(x = 2)$ as an explanation of the conflict does not work, because the procedure can then try $x \leftarrow 3$, and hit another conflict. Clearly, we do not want to learn the infinite sequence $\neg(x = 2),\ \neg(x = 3),\ \neg(x = 4) \ldots$.

Similarly, also a systematic application of the inference rules to enumerate all atoms in a finite basis would be too inefficient. The key point is that the inference rules are applied only to explain conflicts and amend the current partial model, so that the generation of new atoms is *conflict-driven*. This concept is connected with that of *interpolation* (e.g., [83] for interpolation and locality, [23] for a survey on interpolation of ground proofs, and [24] for an approach to interpolation of non-ground proofs): given two inconsistent formulæ A and B, a formula that follows from A and is inconsistent with B is an interpolant of A and B, if it is made only of symbols that appear in both A and B. In a theory $\mathcal{T}$, the notions of being inconsistent and being logical consequence are relative to $\mathcal{T}$, and the interpolant is allowed to contain theory symbols even if they are

not common to A and B. Since an explanation is a formula S' that follows from S and is inconsistent with I, an interpolant of S and I (written as a formula) is an explanation. We illustrate these ideas continuing Example 10:

Example 11. The solution is to observe that $x^2+y^2 \leq 1$ implies $-1 \leq x \wedge x \leq 1$, which is inconsistent with $x = 2$. Note that $-1 \leq x \wedge x \leq 1$ is an interpolant of $x^2 + y^2 \leq 1$ and $x = 2$, as x appears in both. Thus, a desirable explanation is $(x^2+y^2 \leq 1) \Rightarrow x \leq 1$, or $\neg(x^2+y^2 \leq 1) \vee x \leq 1$ in clausal form, which brings the procedure to update the trail to $M = x \geq 2,\ x \geq 1,\ y \geq 1,\ x^2 + y^2 \leq 1,\ x \leq 1$. At this point, $x \geq 2$ and $x \leq 1$ cause another theory conflict, which leads the procedure to learn the lemma $\neg(x \geq 2) \vee \neg(x \leq 1)$. A first step of explanation by resolution between $\neg(x^2 + y^2 \leq 1) \vee x \leq 1$ and $\neg(x \geq 2) \vee \neg(x \leq 1)$ yields $\neg(x^2 + y^2 \leq 1) \vee \neg(x \geq 2)$. A second step of explanation by resolution between $\neg(x^2 + y^2 \leq 1) \vee \neg(x \geq 2)$ and $x \geq 2$ yields $\neg(x^2 + y^2 \leq 1)$, so that the trail is amended to $M = x \geq 2,\ x \geq 1,\ y \geq 1,\ \neg(x^2 + y^2 \leq 1)$, finally repairing the decision (asserting $x^2 + y^2 \leq 1$) that caused the conflict.[4]

In summary, MCsat is a fully model-based procedure, which lifts CDCL to SMT and SMA. Assignments to first-order variables and new literals are involved in decisions, propagations, conflict detections, and explanations, on a par with Boolean assignments and input literals. The theories covered in [35, 55] are the quantifier-free fragments of the theories of equality, linear arithmetic, and boolean values, and their combinations. MCsat is also the name of the implementation of the method as described in [55].

4 Discussion

We surveyed model-based reasoning methods, where inferences build or amend partial models, which guide in turn further inferences, balancing search with inference, and search for a model with search for a proof. We exemplified these concepts for first-order clausal reasoning, and then we lifted them, sort of speak, to theory reasoning. Automated reasoning has made giant strides, and state of the art systems are very sophisticated in working with mostly syntactic information. The challenge of model-based methods is to go towards a semantically-oriented style of reasoning, that may pay off for hard problems or new domains.

Acknowledgments The first author thanks David Plaisted, for starting the research on SGGS and inviting her to join in August 2008; and Leonardo de Moura, for the discussions on MCsat at Microsoft Research in Redmond in June 2013. The third author's work was partially supported by DFG TCRC SFB/TR 14 AVACS (www.avacs.org).

[4] The problem in Examples 10 and 11 appeared in the slides of a talk entitled "Arithmetic and Optimization @ MCsat" presenting joint work by Leonardo de Moura, Dejan Jovanović, and Grant Olney Passmore, and given by Leonardo de Moura at a Schloss Dagsthul Seminar on "Deduction and Arithmetic" in October 2013.

References

1. Ernst Althaus, Evgeny Kruglov, and Christoph Weidenbach. Superposition modulo linear arithmetic SUP(LA). In Silvio Ghilardi and Roberto Sebastiani, editors, *Proceedings of FroCoS-7*, volume 5749 of *Lecture Notes in Artificial Intelligence*, pages 84–99. Springer, 2009.
2. Chandrabose Aravindan and Peter Baumgartner. Theorem proving techniques for view deletion in databases. *Journal of Symbolic Computation*, 29(2):119–148, 2000.
3. Rob Arthan and Paulo Oliva. (Dual) Hoops have unique halving. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Artificial Intelligence*, pages 165–180. Springer, 2013.
4. Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.
5. Leo Bachmair and Harald Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *Journal of the ACM*, 45(6):1007–1049, 1998.
6. Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Refutational theorem proving for hierarchic first-order theories. *Applicable Algebra in Engineering Communication and Computing*, 5:193–212, 1994.
7. Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marjin Heule, Hans Van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, chapter 26, pages 825–886. IOS Press, 2009.
8. Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, and Wolfgang Nejdl. Semantically guided theorem proving for diagnosis applications. In *Proceedings of IJCAI-16*, volume 1, pages 460–465, 1997.
9. Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the model evolution calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.
10. Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper tableaux. In José Júlio Alferes, Luís Moniz Pereira, and Ewa Orłowska, editors, *Proceedings of JELIA-5*, volume 1126 of *Lecture Notes in Artificial Intelligence*, pages 1–17. Springer, 1996.
11. Peter Baumgartner, Ulrich Furbach, and Björn Pelzer. The hyper tableaux calculus with equality and an application to finite model computation. *Journal of Logic and Computation*, 20(1):77–109, 2008.
12. Peter Baumgartner, Björn Pelzer, and Cesare Tinelli. Model evolution calculus with equality – revised and implemented. *Journal of Symbolic Computation*, 47(9):1011–1045, 2012.
13. Peter Baumgartner and Cesare Tinelli. The model evolution calculus as a first-order DPLL method. *Artificial Intelligence*, 172(4–5):591–632, 2008.
14. Peter Baumgartner and Uwe Waldmann. Superposition and model evolution combined. In Renate Schmidt, editor, *Proceedings of CADE-22*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 17–34. Springer, 2009.
15. Peter Baumgartner and Uwe Waldmann. Hierarchic superposition with weak abstraction. In Maria Paola Bonacina, editor, *Proceedings of CADE-24*, volume 7898 of *Lecture Notes in Artificial Intelligence*, pages 39–57. Springer, 2013.
16. Markus Bender, Björn Pelzer, and Claudia Schon. E-KRHyper 1.4: Extensions for unique names and description logic. In Maria Paola Bonacina, editor, *Proceedings*

of CADE-24, volume 7898 of *Lecture Notes in Artificial Intelligence*, pages 126–134. Springer, 2013.
17. Wolfgang Bibel and Peter H. Schmitt, editors. *Automated Deduction - A Basis for Applications (in 2 volumes)*. Kluwer Academic Publishers, 1998.
18. Maria Paola Bonacina. A taxonomy of theorem-proving strategies. In Michael J. Wooldridge and Manuela Veloso, editors, *Artificial Intelligence Today – Recent Trends and Developments*, volume 1600 of *Lecture Notes in Artificial Intelligence*, pages 43–84. Springer, 1999.
19. Maria Paola Bonacina. On theorem proving for program checking – Historical perspective and recent developments. In Maribel Fernández, editor, *Proceedings of PPDP-12*, pages 1–11. ACM Press, 2010.
20. Maria Paola Bonacina and Mnacho Echenim. Theory decision by decomposition. *Journal of Symbolic Computation*, 45(2):229–260, 2010.
21. Maria Paola Bonacina and Jieh Hsiang. Towards a foundation of completion procedures as semidecision procedures. *Theoretical Computer Science*, 146:199–242, 1995.
22. Maria Paola Bonacina and Jieh Hsiang. On semantic resolution with lemmaizing and contraction and a formal treatment of caching. *New Generation Computing*, 16(2):163–200, 1998.
23. Maria Paola Bonacina and Moa Johansson. Interpolation of ground proofs in automated deduction: a survey. *Journal of Automated Reasoning*, 54(4):353–390, 2015.
24. Maria Paola Bonacina and Moa Johansson. On interpolation in automated theorem proving. *Journal of Automated Reasoning*, 54(1):69–97, 2015.
25. Maria Paola Bonacina, Christopher A. Lynch, and Leonardo de Moura. On deciding satisfiability by theorem proving with speculative inferences. *Journal of Automated Reasoning*, 47(2):161–189, 2011.
26. Maria Paola Bonacina and David A. Plaisted. Constraint manipulation in SGGS. In Temur Kutsia and Christophe Ringeissen, editors, *Proceedings of UNIF-28*, RISC Technical Reports, pages 47–54. Johannes Kepler Universität, Linz, 2014.
27. Maria Paola Bonacina and David A. Plaisted. Semantically guided goal-sensitive reasoning: inference system and completeness. In preparation, 2015.
28. Maria Paola Bonacina and David A. Plaisted. Semantically-guided goal-sensitive reasoning: model representation. *Journal of Automated Reasoning*, to appear:29 pages, 2015. Published online 26 June 2015 with DOI: 10.1007/978-3-319-23165-5_8.
29. Maria Paola Bonacina and David A. Plaisted. SGGS theorem proving: an exposition. In Stephan Schulz, Leonardo De Moura, and Boris Konev, editors, *Proceedings of PAAR-4 (2014)*, volume 31 of *EasyChair Proceedings in Computing (EPiC)*, pages 25–38, July 2015.
30. Thierry Boy de la Tour and Mnacho Echenim. Permutative rewriting and unification. *Information and Computation*, 205(4):624–650, 2007.
31. Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.
32. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.
33. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
34. Leonardo de Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.

35. Leonardo de Moura and Dejan Jovanović. A model-constructing satisfiability calculus. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *Proceedings of VMCAI-14*, volume 7737 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2013.
36. Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier, 1990.
37. Santiago Escobar, José Meseguer, and Ralf Sasse. Variant narrowing and equational unification. *Electronic Notes in Theoretical Computer Science*, 238:103–119, 2009.
38. Branden Fitelson. Gibbard's collapse theorem for the indicative conditional: an axiomatic approach. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Artificial Intelligence*, pages 181–188. Springer, 2013.
39. Ulrich Furbach and Claudia Schon. Semantically guided evolution of $\mathcal{SHI}$ ABoxes. In Didier Galmiche and Dominique Larchey-Wendling, editors, *Proceedings of TABLEAUX-22*, volume 8123 of *Lecture Notes in Artificial Intelligence*, pages 17–34. Springer, 2013.
40. Jean H. Gallier and Wayne Snyder. Designing unification procedures using transformations: a survey. *EATCS Bulletin*, 40, 1990.
41. Harald Ganzinger, Viorica Sofronie-Stokkermans, and Uwe Waldmann. Modular proof systems for partial functions with Evans equality. *Information and Computation*, 240(10):1453–1492, 2006.
42. Harald Ganzinger and Uwe Waldmann. Theorem proving in cancellative abelian monoids. In Michael A. McRobbie and John K. Slaney, editors, *Proceedings of CADE-13*, volume 1104 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 1996.
43. Joe Goguen and José Meseguer. Order-sorted unification. *Journal of Symbolic Computation*, 8(4):383–413, 1989.
44. Joe Goguen and José Meseguer. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.
45. Joe Hendrix, Manuel Clavel, and José Meseguer. A sufficient completeness reasoning tool for partial specifications. In Jürgen Giesl, editor, *Proceedings of RTA-16*, volume 3467 of *Lecture Notes in Computer Science*, pages 165–174. Springer, 2005.
46. Joe Hendrix and José Meseguer. Order-sorted equational unification revisited. *Electronic Notes in Theoretical Computer Science*, 290(3):37–50, 2012.
47. Joe Hendrix, José Meseguer, and Hitoshi Ohsaki. A sufficient completeness checker for linear order-sorted specifications modulo axioms. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of IJCAR-3*, volume 4130 of *Lecture Notes in Computer Science*, pages 151–155. Springer, 2006.
48. Jieh Hsiang and Michaël Rusinowitch. Proving refutational completeness of theorem proving strategies: the transfinite semantic tree method. *Journal of the ACM*, 38(3):559–587, 1991.
49. Jieh Hsiang, Michaël Rusinowitch, and Ko Sakai. Complete inference rules for the cancellation laws. In *Proceedings of IJCAI-10*, pages 990–992, 1987.
50. Carsten Ihlemann, Swen Jacobs, and Viorica Sofronie-Stokkermans. On local reasoning in verification. In C. R. Ramakrishnan and Jakob Rehof, editors, *Proceedings of TACAS-14*, volume 4963 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2008.

51. Carsten Ihlemann and Viorica Sofronie-Stokkermans. On hierarchical reasoning in combinations of theories. In Jürgen Giesl and Reiner Hähnle, editors, *Proceedings of IJCAR-5*, volume 6173 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2010.
52. João P. Marques-Silva and Karem A. Sakallah. GRASP: A new search algorithm for satisfiability. In *Proceedings of ICCAD 1996*, pages 220–227, 1997.
53. Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic – Essays in Honor of Alan Robinson*, pages 257–321. MIT Press, 1991.
54. Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986.
55. Dejan Jovanović, Clark Barrett, and Leonardo de Moura. The design and implementation of the model-constructing satisfiability calculus. In Barbara Jobstman and Sandip Ray, editors, *Proceedings of FMCAD-13*. ACM and IEEE, 2013.
56. Deepak Kapur. An automated tool for analyzing completeness of equational specifications. In *Proceedings of ISSTA-94*, pages 28–43. ACM, 1994.
57. Deepak Kapur, Paliath Narendran, Daniel J. Rosenkrantz, and Hantao Zhang. Sufficient-completeness, ground-reducibility and their complexity. *Acta Informatica*, 28(4):311–350, 1991.
58. Shie-Jue Lee and David A. Plaisted. Eliminating duplication with the hyperlinking strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.
59. Vladimir Lifschitz, Leora Morgenstern, and David A. Plaisted. Knowledge representation and classical logic. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, volume 1, pages 3–88. Elsevier, 2008.
60. Sharad Malik and Lintao Zhang. Boolean satisfiability: from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76–82, 2009.
61. William W. McCune. OTTER 3.3 reference manual. Technical Report ANL/MCS-TM-263, MCS Division, Argonne National Laboratory, Argonne, IL, USA, 2003.
62. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In David Blaauw and Luciano Lavagno, editors, *Proceedings of DAC-39*, pages 530–535, 2001.
63. Greg Nelson. Combining satisfiability procedures by equality sharing. In Woodrow W. Bledsoe and Don W. Loveland, editors, *Automatic Theorem Proving: After 25 Years*, pages 201–211. American Mathematical Society, 1983.
64. Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages*, 1(2):245–257, 1979.
65. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.
66. Gerald E. Peterson and Mark E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981.
67. David A. Plaisted. Mechanical theorem proving. In Ranan B. Banerji, editor, *Formal Techniques in Artificial Intelligence*, pages 269–320. Elsevier, 1990.
68. David A. Plaisted. Equational reasoning and term rewriting systems. In Dov M. Gabbay, C. J. Hogger, and John Alan Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume I: Logical Foundations, pages 273–364. Oxford University Press, 1993.
69. David A. Plaisted. Automated theorem proving. *Wiley Interdisciplinary Reviews: Cognitive Science*, 5(2):115–128, 2014.

70. David A. Plaisted and Yunshan Zhu. *The Efficiency of Theorem Proving Strategies*. Friedrich Vieweg & Sohns, 1997.
71. David A. Plaisted and Yunshan Zhu. Ordered semantic hyper linking. *Journal of Automated Reasoning*, 25:167–217, 2000.
72. Gordon Plotkin. Building in equational theories. *Machine Intelligence*, 7:73–90, 1972.
73. George A. Robinson and Lawrence Wos. Paramodulation and theorem proving in first order theories with equality. *Machine Intelligence*, 4:135–150, 1969.
74. John Alan Robinson. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1:227–234, 1965.
75. John Alan Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.
76. John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
77. Michaël Rusinowitch. Theorem-proving with resolution and superposition. *Journal of Symbolic Computation*, 11(1 & 2):21–50, 1991.
78. Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A highly efficient OWL reasoner. In Alan Ruttenberg, Ulrike Sattler, and Cathy Dolbear, editors, *Proceedings of OWLED-5*, volume 432 of *CEUR*, 2008.
79. James R. Slagle. Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 14(4):687–697, 1967.
80. John Slaney, Ewing Lusk, and William McCune. SCOTT: Semantically constrained Otter. In Alan Bundy, editor, *Proceedings of CADE-12*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 764–768. Springer, 1994.
81. Viorica Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In Robert Nieuwenhuis, editor, *Proceedings of CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2005.
82. Viorica Sofronie-Stokkermans. Hierarchical and modular reasoning in complex theories: The case of local theory extensions. In Boris Konev and Frank Wolter, editors, *Proceedings of FroCoS-6*, volume 4720 of *Lecture Notes in Computer Science*, pages 47–71. Springer, 2007.
83. Viorica Sofronie-Stokkermans. Interpolation in local theory extensions. *Logical Methods in Computer Science*, 4(4):Paper 1, 2008.
84. Jürgen Stuber. Superposition theorem proving for abelian groups represented as integer modules. *Theoretical Computer Science*, 208(1-2):149–177, 1998.
85. Uwe Waldmann. Superposition for divisible torsion-free abelian groups. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of CADE-15*, volume 1421 of *Lecture Notes in Computer Science*, pages 144–159. Springer, 1998.
86. Larry Wos, D. Carson, and G. Robinson. Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the ACM*, 12:536–541, 1965.
87. Hantao Zhang and Jian Zhang. MACE4 and SEM: A comparison of finite model generators. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Artificial Intelligence*, pages 101–130. Springer, 2013.
88. Lintao Zhang and Sharad Malik. The quest for efficient boolean satisfiability solvers. In Andrei Voronkov, editor, *Proceedings of CADE-18*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 295–313. Springer, 2002.