Journal of Automated Reasoning article No.
DOI 10.1007/s10817-015-9334-4 in 56(2):113–141, February 2016

Semantically-Guided Goal-Sensitive Reasoning: Model Representation

Maria Paola Bonacina · **David A. Plaisted**

Received: 13 April 2015 / Accepted: 8 June 2015 / Published online: 26 June 2015

Dedicated to the memory of
Mark E. Stickel,
friend and colleague.

Abstract SGGS (*Semantically-Guided Goal-Sensitive* reasoning) is a clausal theorem-proving method, which generalizes to first-order logic the Davis-Putnam-Loveland-Logemann procedure with *conflict-driven clause learning* (DPLL-CDCL). SGGS starts from an *initial interpretation*, and works towards modifying it into a model of a given set of clauses, reporting unsatisfiability if there is no model. The state of the search for a model is described by a structure, called *SGGS clause sequence*. We present SGGS clause sequences as a formalism to represent models; and we prove their properties related to the mechanisms of SGGS for *clausal propagation*, *conflict solving*, and *conflict-driven model repair* at the first-order level.

Keywords Theorem Proving · Model Building · Semantic Guidance

Prologue: Mark Stickel, A Recollection, by Maria Paola Bonacina

I had the pleasure of meeting with Mark Stickel at numerous CADE conferences, possibly as early as CADE-8 in Oxford, when I had just completed my undergraduate studies, and Mark had a paper on his Prolog Technology Theorem Prover, and as late as CADE-22 in Montréal, when we had tea together on the boat cruising the St. Lawrence river during the excursion. Mark had a quiet and reserved personality, kept words to a minimum, but was very kind.

Maria Paola Bonacina
Dipartimento di Informatica, Università degli Studi di Verona, Strada Le Grazie 15, I-37134 Verona, Italy
E-mail: mariapaola.bonacina@univr.it

David A. Plaisted
Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, USA
E-mail: plaisted@cs.unc.edu

Mark and I got to know each other better when I was on research assignment from The University of Iowa, and I visited the Artificial Intelligence Center of SRI International, where Mark was a Principal Scientist. Mark and I discussed theorem proving and artificial intelligence every day: it was a great time, and a true learning experience. After that visit, I went back to The University of Iowa full of enthusiasm, and wrote right away my successful NSF CAREER award proposal.

Herbrand Award nominations are confidential, but now that Mark, sadly, has left us prematurely, I suppose I can write that I had the pleasure and honor of nominating him for the Herbrand Award. I was proud and happy when my nomination of Mark went through right away at the first attempt, undoubtedly a sign of how much Mark deserved the award in the eyes of all colleagues involved.

When Bill McCune unexpectedly passed away, and I had the idea of a volume to honor his memory, I immediately thought of Mark Stickel as co-editor. I was delighted that Mark accepted. Although he liked to minimize it, his contribution was precious. He found some excellent reviewers, and he was especially keen on encouraging papers about applications of automated reasoning to mathematics, knowing how much Bill cared for that topic. While being already terminally ill, Mark had the patience of reading again all final versions of the papers, and let me know of any error that we could fix. The letter with which Mark informed me that he had only a few more months to live was admirable for calm, quiet rationality, and matter-of-fact style. I admired his dignity, while being immensely sad upon reading it. Mark would have loved to see the book in Bill's memory before passing. I was told that his copy of the book arrived a couple of days later. It is just one more reason for contributing to a volume in his memory, that may help to keep his intellectual legacy alive.

1 Introduction

Mark Stickel received the Herbrand Award at the Eighteenth International Conference on Automated Deduction (CADE), held during the Federated Logic Conference (FLoC) of 2002. During that FLoC, Natarajan Shankar gave a brilliant invited talk on "little engines of proof" [83], meaning decision procedures for satisfiability in decidable fragments of first-order theories, to be embedded in proof assistants or verification tools. In his memorable Herbrand Award acceptance speech, Mark Stickel upheld resolution, and in general "big engines of proof," meaning semi-decision procedures for validity in first-order logic, and suggested the importance of combining their strengths with those of reasoning at the propositional, or ground, level.

Several of Mark Stickel's outstanding contributions to automated reasoning can be seen retrospectively as pioneering steps of this kind. For instance, his work on *AC-unification* [86,75] and *theory resolution* [88] was about building theories into engines for first-order theorem proving. He developed the first-order theorem provers SNARK, based on resolution and paramodulation, and PTTP (Prolog Technology Theorem Prover) [87,89,90], based on model elimination, but also an implementation of the Davis-Putnam-Loveland-Logemann (DPLL) procedure [30,29], named LDPP (List-based Davis-Putnam Prover), which was one of the early milestones of the DPLL renaissance for propositional satisfiability.

Because they apply in domains where satisfiability is decidable, decision procedures search for a *model*, rather than a proof, and produce a proof if they find that no model exists. Because in first-order logic theorem proving is semi-decidable, and model finding is not, theorem-proving strategies search for a *proof*. The dichotomy between decision procedures and semi-decision procedures is also a dichotomy between *solving* and *proving*, or between models and proofs. However, also in first-order logic there is interest in finding models (cf. [23] for a systematic treatment), in using interpretations for *semantic guidance* (e.g., [81,94,84,70] and Section 2.6 of [14] for a survey), and there are theorem-proving methods based on *model elimination* (cf. [68] for the original formulation with chains of literals and [63,8,65] for surveys of the formulation with tableaux), or on *model evolution* [10], that find refutations by excluding all models, which requires ways to represent them.

SGGS, from *Semantically-Guided Goal-Sensitive* reasoning, is a new theorem-proving method, which brings to the first-order level the *model-based* style of DPLL with *conflict-driven clause learning* (DPLL-CDCL) [49,72,96,69]. SGGS is based on an approach to the *representation of first-order models*, which is the topic of this article, and whose basic ideas we outline here starting from DPLL-CDCL.

In DPLL-CDCL the current partial candidate model is represented by a sequence of propositional literals, called *trail*, meaning that the literals in the sequence are true in the model. A literal L in the trail can be either *decided*, as the procedure decided to make it true by a mere guess, or *implied*, as making it true is the only way to satisfy a clause whose other literals are all false, because their complements are in the trail: such a clause is called the *justification* of the implied literal. A clause whose literals are all false, because their complements are in the trail, is a *conflict clause*. Detecting implied literals and conflict clauses is the objective of *clausal* or *Boolean constraint propagation*, which is a main activity of the procedure, that resorts to a decision only when no propagation is possible. The principle of propagation is that if literal L is in the trail, all occurrences of $\neg L$ are false. Propagation of truth values is simple in propositional logic, because truth and falsity are *symmetric*: if L is true, $\neg L$ is false, and vice versa. In first-order clausal logic the (implicit) presence of quantifiers destroys the symmetry: if L is true, all its ground instances are true, all ground instances of $\neg L$ are false, and $\neg L$ is false; but if $\neg L$ is false, all we know is that there exists a ground instance L' of L such that $\neg L'$ is false and L' is true. SGGS *restores the symmetry* by introducing the notion of *uniform falsity*: a literal is *uniformly false* in an interpretation, if all its ground instances are false in that interpretation. Then, if L is true, $\neg L$ is uniformly false, and vice versa.

SGGS works with a set S of first-order clauses to be refuted, and a given initial interpretation I for semantic guidance. The trail of literals of DPLL-CDCL is replaced by a sequence of *constrained clauses* with *selected literals*, called *SGGS clause sequence*: every literal in an SGGS clause sequence is required to be either true in I or uniformly false in I, so that its occurrence in the sequence tells the truth value in I of *all* its ground instances that satisfy its constraints. SGGS then specifies how an SGGS clause sequence Γ *induces an interpretation* $I[\Gamma]$ different from I. First-order propagation is represented by an *assignment function*, which maps a selected literal M to a selected literal L in Γ, if M is uniformly false in $I[\Gamma]$ due to the occurrence of L in Γ, so that M stands to L, like $\neg L$ stands to L at the propositional level. With

uniform falsity and first-order propagation, we can generalize to the first-order level the propositional notions of *conflict clause*, *implied literal*, and *justification*: a clause is in conflict with $I[\Gamma]$, if all its literals are uniformly false in $I[\Gamma]$; a literal is *implied*, if all other literals in its clause are uniformly false in $I[\Gamma]$, and the clause is its *justification*. Upon this basis, the SGGS inference system uses *instance generation* to build SGGS clause sequences, *resolution* to *explain conflicts*, and a mechanism à la CDCL to *solve conflicts*, by modifying Γ so that $I[\Gamma]$ satisfies the conflict clause, except when the conflict clause is empty, which signals the discovery of a refutation. The SGGS inference system is the topic of a twin article [21], where we prove its *refutational completeness* and *goal sensitivity*.

This article is organized as follows. Section 2 surveys recent developments of the idea of integrating *search* of a model and *inference* towards a proof, a main source of inspiration for SGGS. Section 3 introduces *constrained clauses*, *uniform falsity*, *SGGS clause sequences*, and *SGGS-derivations*. Section 4 describes how an SGGS clause sequence Γ induces an interpretation $I[\Gamma]$. Section 5 defines *first-order propagation* and *assignment functions*. Sections 6 and 7 show how clauses in an SGGS clause sequence Γ can be *conflict clauses*, *justifications* of *implied literals*, or clauses that make $I[\Gamma]$ different from I; and how Γ entails a formula encoding $I[\Gamma]$ and other interpretations yet to be considered. To make this article self-contained with respect to [21], we intersperse descriptions of SGGS inference mechanisms with the presentation of model representation features, illustrating their connections. A comparison with related work and a discussion close the article. An exposition of SGGS and its constraint solving part appeared in preliminary form in [20] and [19], respectively.

2 Proofs and Models, Inference and Search

Proofs and models are the mainstay of automated reasoning. Traditionally, proofs have taken center stage, for several reasons, both historical and technical. The centrality of the concept of proof dates back to Hilbert's program at the dawn of the XX century, and has affected the development of mathematics, logic, and computer science since then [28]. The completeness and expressivity of first-order logic have made it the logic of choice for much of automated reasoning and artificial intelligence. However, in first-order logic, theorem proving, or proof finding, is semi-decidable, while model finding is not, and therefore the focus on first-order logic has reinforced the emphasis on proofs over models.

The growth of algorithmic reasoning and of its applications to software has changed the balance, because models are useful for applications and intuitive for users. An assignment to program variables, which describes a possible state of a program's execution, is a model from a logical point of view. Thus, for instance, models become "moles" to exercise program paths in testing or examples for example-driven synthesis, and a reasoner that generates models supports automated test generation and program synthesis (e.g., [33,62] and [16] for a survey and more references).

Algorithmic reasoning applies when satisfiability is decidable, and one can get an algorithm, a decision procedure, to decide it (e.g., [39,22,58] for general sources). The archetype for algorithmic reasoning is propositional logic and its satisfiability

problem, known as SAT. Contemporary SAT-solvers are based on the DPLL-CDCL procedure. The original Davis-Putnam (DP) procedure [30] featured propositional resolution, which incurs duplication of literals across clauses (cf. the analysis in [78]), and is unnecessarily non-deterministic for propositional logic. The DPLL procedure [29] replaced propositional resolution with *splitting*, understood first as breaking disjunctions apart by *case analysis*, and then as *guessing*, or *deciding*, the truth value of a propositional variable, in order to build a model of the given set of clauses.

This replacement made the fortune of the DPLL procedure for propositional logic, because it avoids the growth of clauses and the non-determinism of resolution. It opened the way to the study of efficient techniques for *clausal* or *Boolean constraint propagation* (e.g., [95,96,37,36,48]), where Mark Stickel was a pioneer, and to the reading of DPLL as a *model-based* procedure, which *searches* for a model, with all its operations centered around the candidate partial model.

It was perceived early on [91] that a better balance between *guessing* and *reasoning*, or *search* and *inference*, was to be sought. This led to DPLL-CDCL [49,72], where propositional resolution comes back as a mechanism to generate *lemmas*. The model-based character of the procedure becomes 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 model is repaired in such a way to remove the conflict, satisfy the lemma, and backjump as far away as possible from the conflict. An early forerunner of DPLL-CDCL was the *semantic tree method with lemma generation* (cf. pages 284–288 of [76]). Nowadays, DPLL-CDCL is considered a different procedure from DPLL.

The same trend emerges in *satisfiability modulo theories* (SMT) (cf. [2,34] for surveys). A basic paradigm for SMT is given by the DPLL($\mathscr{T}$) procedure [74], which integrates in DPLL a decision procedure for satisfiability in the quantifier-free fragment of a first-order theory $\mathscr{T}$. However, the CDCL mechanism in DPLL($\mathscr{T}$) is still by propositional resolution. A current frontier in research on decision procedures and SMT-solving is to find ways to do for first-order theories what resolution does for propositional logic, that is, *explain* why a theory conflict occurred, and deduce theory lemmas in an efficient conflict-driven fashion. This problem was studied in several existential fragments of arithmetic [71,56,27,51,52,44]. A paradigm that generalizes CDCL to any theory that can be equipped with clausal inference rules to explain conflicts was presented in [35,50], under the name of *MC-sat*, for *model-constructing satisfiability procedures*. The crux is that in order to explain conflicts in a first-order theory the procedure needs to generate new atoms, unlike in the propositional case, where all explanations are built from input atoms. Thus, in order to preserve termination, it is necessary to show that all atoms come from a *finite basis*.

Generalizing features of DPLL is an objective also in first-order theorem proving. Resolution and superposition behave very well on Horn problems, but perform less well with very long non-Horn clauses, including the ground ones that may arise in applications, and that SMT-solvers handle by splitting. One line of research investigated ways of *splitting* first-order clauses into disjunctions that do not share variables (e.g., [92,80,41,45,40]). These approaches capture the view of DPLL splitting as a way to decompose clauses, rather than as a guess of an assignment to a literal, in keeping with the fact that resolution and superposition do not work on an explicit representa-

tion of a model. In these inference systems models remain implicit, and come to the forefront only in their proofs of refutational completeness, using *transfinite semantic trees* to survey models and show that the inference system excludes them all [46], or the *rewrite model* to show that a set saturated by the inference system has a model if it does not contain the empty clause [1]. Another line of research proposed a *modular* integration of DPLL($\mathscr{T}$) and superposition, where the SMT-solver deals with ground clauses and *interpreted*, or *defined*, symbols, and the superposition-based engine with non-ground clauses and *uninterpreted*, or *free*, symbols [32,17,18]. This approach is *model-based* and coherent with the view of DPLL splitting as a guess of an assignment, because it lets superposition use literals in the trail of DPLL($\mathscr{T}$) as premises. However, the CDCL mechanism remains propositional as in DPLL($\mathscr{T}$).

Instance-based first-order theorem-proving methods are connected with propositional, or ground, satisfiability testing, since their origin from the Herbrand theorem (e.g., [24]), which states that for all unsatisfiable sets S of clauses there exists a finite unsatisfiable set S' of ground instances of clauses in S. Instance-based methods search for S', by generating sets of ground instances of clauses in S, and testing them for satisfiability. The first such procedure was *Gilmore method*, which used enumeration of ground instances for instance generation and the multiplication method for propositional satisfiability (e.g., [24]). Since both were inefficient, the idea was abandoned until *hyperlinking* [59–61], which used DPLL for SAT testing and hyperlinks for instance generation (e.g., it applies the most general unifier of a hyperresolution step to generate instances of the parents rather than the hyperresolvent).

Early instance-based methods had a generate-and-test flavor, which was replaced over time by the notion of *model-driven* instance generation: the procedure starts with a candidate model of S, generates ground instances to exclude it, updates it to satisfy them, and continues until either a model is found, or unsatisfiability is proved. The origin of this shift of perspective may be traced back to the first successor of hyperlinking CLINS-S [25,26], the *disconnection calculus* [13,64,66,67], *ordered semantic hyperlinking* (OSHL) [79,77], and neighbor tableaux-based methods such as *hypertableaux* [3,12] (cf. Section 7.3 of [15] for a discussion of methods that hybridize instantiation and tableaux). The drive to generalize features of DPLL to instance-based methods gave rise to FDPLL [4], for *First-order DPLL*, its successor the *model-evolution calculus* (MEC) [10,5,6,11,9], and the *Inst-Gen method* [42, 43,55,54]. Theorem provers embodying the instance generation principle perform especially well in *effectively propositional logic* [31,73,38].

In [47,53] these methods were classified as *saturation-based* methods, such as hyperlinking and Inst-Gen, where the interleaving of instance generation and ground reasoning is *coarse-grained*, or *tableaux-based* methods, such as disconnection and model-evolution calculi, where the interleaving is *fine-grained*. Coarse-grained interleaving facilitates the integration of SAT or SMT-solvers taken "off the shelf." The model is built by the solver and communicated to the first-order part. For example, Inst-Gen employs *literal selection* to restrict instance generation rules similar to hyperlinking, and information on which ground literals are in the trail of the solver is relevant for literal selection [54].

Fine-grained interleaving leads to methods that are model-based at the first-order level. For example, the model evolution calculus (MEC) lifts splitting to first-order

logic: in DPLL, splitting assigns *true* to a propositional variable A on one branch, and *false* on the other; MEC asserts for instance $A(x)$, meaning $\forall x\ A(x)$, on one branch, and $\neg A(c)$, the Skolemized form of $\neg\forall x\ A(x) \equiv \exists x\ \neg A(x)$, on the other. Since Skolemization changes the signature, and Skolem constants, being new, do not unify with other non-variable terms, MEC employs *parameters*, rather than Skolem constants, in a way reminiscent of the δ-rule to eliminate existential quantifiers in tableaux (cf. [85]). Asserted literals are stored in a *context* Λ, which represents a Herbrand interpretation I_Λ, seen as a candidate partial model of the input set of clauses S: MEC determines whether $I_\Lambda \models L$, for L an atom in the Herbrand base of S, by considering the most specific literal in Λ that subsumes L, picking L with positive sign in case of a tie. If I_Λ is not a model of S, MEC unifies input clauses against Λ to find potentially falsified instances, that are subject to splitting, to modify Λ and repair I_Λ. Otherwise, the system recognizes that Λ cannot be fixed and declares S unsatisfiable.

This excursus shows how intertwining *search* for a model and *inference* towards either fixing the model or finding a refutation is a main trend at the frontier of reasoning research. SGGS advances this quest, by *lifting DPLL-CDCL to first-order logic*, somewhat like MC-sat does for decidable fragments of first-order theories. If MEC emphasizes splitting, SGGS emphasizes *propagation*, and *explanation* and *solution of conflicts*. In the following we develop the SGGS approach to model representation that makes this possible.

3 SGGS Clause Sequences

We assume standard notions and notations in logic and reasoning, including those about terms, atoms, literals, clauses, substitutions, and instances. The symbol $\equiv$ is syntactic identity, and $top(t)$ denotes the top symbol of term t. If L is a literal, $at(L)$ is its atom, and if T is a set of literals, $at(T) = \{at(L) :\ L \in T\}$. The *Herbrand universe* of a set S of clauses is the set of ground terms made of function and constant symbols from S, with a new constant symbol added if S has none. A *Herbrand interpretation I over S* has the Herbrand universe of S as domain, and interprets constant and function symbols so that $t^I = t$ for all ground terms t, where t^I is the interpretation of t in I. A *Herbrand substitution* over S for a formula C replaces all variables in C by elements of the Herbrand universe of S. The set of Herbrand substitutions over S for C is written $H_S(C)$. The notation $\models_{H,S} X$ means that for all Herbrand interpretations I over S, $I \models X$. Since in clausal theorem proving it suffices to consider Herbrand interpretations, we omit the subscripts S and H, and write $\models$ for $\models_{H,S}$.

In propositional logic a model of a set of clauses, or a partial model if not all literals are specified, can be represented by a sequence of literals. For first-order logic SGGS uses a sequence of *clauses with constraints* and *selected literals*. In this section we define these sequences and their ingredients. We begin with constraints:

Definition 1 (Constraint) An *atomic constraint* is either *true* or *false* or an expression of the form $t \equiv u$ or $top(t) = f$, where t and u are terms, and f is a function symbol. An atomic constraint is a constraint, the negation, conjunction, or disjunction of constraints is a constraint, and nothing else is a constraint.

Constraints are only meaningful for Herbrand interpretations: $\models t \equiv u$ for ground terms t and u if and only if t and u are the same element of the Herbrand universe of S, and $\models top(t) = f$ if the top symbol of ground term t is f. Thus, either $\models A\vartheta$ or $\models \neg A\vartheta$ for constraint A and $\vartheta \in H_S(A)$.

A constraint is in *standard form*, if it is either *true* or *false* or a conjunction of distinct atomic constraints of the form $x \not\equiv y$ and $top(x) \neq f$, where x and y are variables. A constraint $top(x) \neq f$ says that x cannot be replaced by a term whose top function symbol is f, while a constraint $x \not\equiv y$ specifies that x and y may not be replaced by identical terms. In this article all constraints are in standard form: standardization is covered in [19].

Definition 2 (Constrained clause) A *constrained clause* is a formula $A \rhd C$, read "C under A" or "C if A," where A is a constraint and C is a clause. Any variable that appears in A and not in C is implicitly existentially quantified.

We write $A \rhd C[L]$ to say that literal L is *selected* in C; we call $A \rhd L$ a *constrained literal*; and if L is the selected literal of C, $L\vartheta$ is the selected literal of $C\vartheta$. Constraint and selected literal may be omitted when clear from context, and $true \rhd C$ is usually abbreviated as C.

Just like a clause represents its ground instances, a constrained clause $A \rhd C$ represents its *constrained ground instances*, that are the ground instances of C that satisfy the constraint A:

Definition 3 (Constrained ground instances) Given a set S of clauses, the set of *constrained ground instances* (cgi) of $A \rhd C$ is $Gr(A \rhd C) = \{C\vartheta : \models A\vartheta,\ \vartheta \in H_S(C)\}$. If L is the selected literal of C, then $Gr(A \rhd L) = \{L\vartheta : C\vartheta \in Gr(A \rhd C)\}$, and $\neg Gr(A \rhd L)$ is $Gr(A \rhd \neg L)$.

Note how $Gr(true \rhd C)$ contains all ground instances of C, while $Gr(false \rhd C) = \emptyset$. If the constraint is understood, $Gr(A \rhd C)$ and $Gr(A \rhd L)$ can be written $Gr(C)$ and $Gr(L)$, respectively.

Example 1 For a clause $x \not\equiv y \rhd P(x,y)$, $P(a,b) \in Gr(x \not\equiv y \rhd P(x,y))$, but $P(b,b) \notin Gr(x \not\equiv y \rhd P(x,y))$.

We consider next the satisfaction of constrained clauses and literals. Let J be an interpretation. We have $J \models Gr(A \rhd L)$, if $J \models M$ for all $M \in Gr(A \rhd L)$; $J \models \neg Gr(A \rhd L)$, if $J \models M$ for all $M \in \neg Gr(A \rhd L)$; and $J \models Gr(A \rhd C)$, if $J \models D$ for all $D \in Gr(A \rhd C)$. Since variables in clauses, and their literals, are implicitly universally quantified, $J \models A \rhd L$, if $J \models Gr(A \rhd L)$; and $J \models A \rhd C$, if $J \models Gr(A \rhd C)$.

A key point in the SGGS approach to model representation is to reproduce at the first-order level the symmetry between truth and falsity. That is, we want a notion of falsity that holds for a literal L, when $\neg L$ is true. This leads us to say that L is *uniformly false* in J, if all its ground instances are false in J, which is indeed equivalent to $\neg L$ being true in J. Similarly, for clauses, we say that a clause C is *uniformly false* in J, if all its literals are, which means all ground instances of C are false in J, and $\neg C$ is true in J. In symbols, and for constrained clauses, we have:

Definition 4 (Uniform falsity) Given interpretation J and clause $A \rhd C$, a literal L of $A \rhd C$ is *uniformly false* in J, if $J \models \neg Gr(A \rhd L)$; and $A \rhd C$ is *uniformly false* in J, if $J \models \neg Gr(A \rhd L)$ for all literals L of C.

Uniform falsity is stronger than falsity, because in order to falsify a literal or clause, it suffices to falsify one ground instance, whereas uniform falsity says that *all* ground instances are false. If $A \rhd C$ is uniformly false in J, we say that $A \rhd C$ is a *conflict clause* with respect to J, or $A \rhd C$ is *in conflict* with J.

For *semantic guidance*, SGGS assumes a fixed *initial interpretation* I. By applying uniform falsity to I, we say that a constrained literal $A \rhd L$ is *I-false*, if it is uniformly false in I. Note that being I-false is stronger than being false in I. In order to make the terminology symmetric, we also say that $A \rhd L$ is *I-true*, if it is true in I, that is, if $I \models A \rhd L$. Of course, a constrained literal can be neither I-true nor I-false. Given a constrained clause C, its sets of I-true and I-false literals are denoted $tlits(C)$ and $flits(C)$, respectively.

For clauses, $A \rhd C$ is *I-all-true*, if all its literals are I-true, and *I-all-false* if all its literals are I-false. Being I-all-true is stronger than being true in I: for $A \rhd C$ to be true in I it suffices that for all its constrained ground instances $C\vartheta \in Gr(A \rhd C)$ there is a literal $L\vartheta$ of $C\vartheta$ that is true in I; whereas being I-all-true means that for all its constrained ground instances $C\vartheta \in Gr(A \rhd C)$ all literals $L\vartheta$ of $C\vartheta$ are true in I. Similarly, being I-all-false is stronger than being false in I: for $A \rhd C$ to be false in I it suffices that there is a constrained ground instance $C\vartheta \in Gr(A \rhd C)$ such that all literals $L\vartheta$ of $C\vartheta$ are false in I; whereas being I-all-false means that for all its constrained ground instances $C\vartheta \in Gr(A \rhd C)$ all literals $L\vartheta$ of $C\vartheta$ are false in I. Being I-all-false is the same as being in conflict with I. We tolerate this redundancy in the terminology for the sake of symmetry. I-all-true and I-all-false are used only for the given fixed initial interpretation I. We have all the elements to define *SGGS clause sequences*:

Definition 5 (SGGS clause sequence) An *SGGS clause sequence* is a possibly empty, finite sequence of constrained clauses $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$ ($n \geq 0$) such that for all j, $1 \leq j \leq n$:

1. $C_j = flits(C_j) \uplus tlits(C_j)$: every literal is either I-true or I-false;
2. If $flits(C_j) \neq \emptyset$, then $L_j \in flits(C_j)$: if a clause has I-false literals, one is selected.

The length of Γ is $|\Gamma| = n$. For all j, $1 \leq j \leq |\Gamma|$, clause $A_j \rhd C_j[L_j]$ has *index* j in Γ, and $\Gamma|_j = A_1 \rhd C_1[L_1], \ldots, A_j \rhd C_j[L_j]$ is the *prefix* of Γ of length j.

The empty sequence is denoted by ε; and by convention $\Gamma|_0 = \varepsilon$. The requirement that every literal is either I-true or I-false ensures that every literal in the sequence tells the truth value in I of all its constrained ground instances. Since I-false literals are preferred for selection, an I-true literal is selected only in an I-all-true clause.

Given a set S of first-order clauses to be refuted and the initial interpretation I, an *SGGS-derivation* is a series $\Gamma_0 \vdash \Gamma_1 \vdash \ldots \Gamma_j \vdash \ldots$ of SGGS clause sequences, where $\Gamma_0 = \varepsilon$, and each Γ_j with $j > 0$ is generated from Γ_{j-1}, S, and I, by an SGGS inference rule. For example, the main inference rule, *SGGS-extension*, adds to the sequence an instance (not necessarily ground) of an input clause. Whenever an SGGS inference

adds clauses to a sequence, the new clauses are added to the rightmost end of the sequence or to the left of some other clause, but never to the leftmost end of the sequence. Thus, SGGS clause sequences basically grow from left to right.

An SGGS-derivation represents a search for a model of S, and it is a *refutation* if there is a $j > 0$, such that Γ_j contains the empty clause, written $\bot$, which signals that no model can be found. SGGS searches for a model distinct from I, because $I \not\models S$ (otherwise, the problem is trivially solved upfront). This is the reason why I-false literals are preferred for selection: selected literals play a major rôle in determining the interpretation $I[\Gamma]$ induced by Γ, and selecting I-false literals makes $I[\Gamma]$ different from I, as we see in the next section.

4 Interpretations Induced by SGGS Clause Sequences

In this section we learn how SGGS clause sequences represent interpretations. We begin with three key concepts: the *partial interpretation* $I^p(\Gamma)$ induced by an SGGS clause sequence Γ, the *proper constrained ground instances*, and the *complementary constrained ground instances* of a selected literal in Γ. The former are those that the literal contributes to $I^p(\Gamma)$, while the latter are those that cannot be added to $I^p(\Gamma)$, because otherwise it would become inconsistent. A clause in Γ whose selected literal has neither proper nor complementary constrained ground instances is *disposable*: such a clause is satisfied by the partial interpretation induced by the clauses of smaller index in the sequence, so that it does not add anything and can be removed. By resorting to the initial interpretation I to determine the truth of ground literals left unspecified by $I^p(\Gamma)$, we define the interpretation $I[\Gamma]$ induced by Γ.

The *partial interpretation induced* by $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$ is defined inductively over the length of the sequence: $I^p(\Gamma)$ will be given by $I^p(\Gamma|_{n-1})$ plus those constrained ground instances of $A_n \rhd L_n$ that can be added to satisfy ground instances of $A_n \rhd C_n[L_n]$ not satisfied by $I^p(\Gamma|_{n-1})$. These constrained ground instances are called *proper*. Thus, partial interpretations and proper constrained ground instances are mutually defined as follows:

Definition 6 (Induced partial interpretation) The *partial interpretation induced* by $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$, denoted $I^p(\Gamma)$, is defined inductively as follows:

- If $\Gamma = \varepsilon$, then $I^p(\Gamma) = \emptyset$;
- Otherwise, $I^p(\Gamma) = I^p(\Gamma|_{n-1}) \cup pcgi(A_n \rhd L_n, \Gamma)$.

A constrained ground instance $C[L]$ of $A_j \rhd C_j[L_j]$ is *proper* if it is not satisfied by $I^p(\Gamma|_{j-1})$, and its selected literal L does not appear negated in $I^p(\Gamma|_{j-1})$, so that L can be added to $I^p(\Gamma|_{j-1})$ to form $I^p(\Gamma|_j)$ and satisfy $C[L]$:

Definition 7 (Proper constrained ground instances) For $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$, for all j, $1 \leq j \leq n$, the set of *proper constrained ground instances (pcgi)* of clause $A_j \rhd C_j[L_j]$, and of its selected literal $A_j \rhd L_j$, is defined as follows:

- $pcgi(A_j \rhd C_j[L_j], \Gamma) = \{C[L] \in Gr(A_j \rhd C_j[L_j]) : I^p(\Gamma|_{j-1}) \cap C[L] = \emptyset \wedge \neg L \notin I^p(\Gamma|_{j-1})\}$: $C[L]$ is not satisfied by $I^p(\Gamma|_{j-1})$, because none of its literals is in $I^p(\Gamma|_{j-1})$, and can be satisfied by adding L, because $\neg L$ is not in $I^p(\Gamma|_{j-1})$;

- $pcgi(A_j \rhd L_j, \Gamma) = \{L : \ C[L] \in pcgi(A_j \rhd C_j[L_j], \Gamma)\}$: that is, the selected literals from the *pcgi*'s of the clause.

Then we can define the set of proper constrained ground instances of Γ itself as $pcgi(\Gamma) = \bigcup_{j=1}^{n} pcgi(A_j \rhd L_j, \Gamma)$ so that $pcgi(\Gamma) = I^p(\Gamma)$.

Example 2 If Γ is the sequence of unit clauses $P(a,x), P(b,y), \neg P(z,z), P(u,v)$, then: all ground instances of $P(a,x)$ are proper; all ground instances of $P(b,y)$ are proper; all ground instances of $\neg P(z,z)$ are proper, except $\neg P(a,a)$ and $\neg P(b,b)$; all ground instances of $P(u,v)$ are proper, except those of the form $P(a,t)$, $P(b,t)$, and $P(t,t)$ for any ground term t.

An SGGS clause sequence will typically grow by SGGS-extension inferences. SGGS-extension looks for instances of clauses in S that are not satisfied by $I^p(\Gamma)$; if it finds any, it appends it to Γ to obtain a Γ', such that $I^p(\Gamma')$ satisfies it. Thus, intuitively, Γ will have at index j a clause $A_j \rhd C_j[L_j]$ with some constrained ground instance C such that $I^p(\Gamma|_{j-1}) \cap C = \emptyset$: clause $A_j \rhd C_j[L_j]$ was added precisely to satisfy such instances, and this is why they are deemed *proper*.

A constrained ground instance of the clause at index j in Γ is *complementary*, if it is not satisfied by $I^p(\Gamma|_{j-1})$ and cannot be satisfied by adding its selected literal, because its selected literal appears negated in $I^p(\Gamma|_{j-1})$:

Definition 8 (Complementary constrained ground instances) For $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$, for all j, $1 \leq j \leq n$, the set of *complementary constrained ground instances (ccgi)* of clause $A_j \rhd C_j[L_j]$ and of its selected literal $A_j \rhd L_j$ is defined as follows:

- $ccgi(A_j \rhd C_j[L_j], \Gamma) = \{C[L] \in Gr(A_j \rhd C_j[L_j]) : \ I^p(\Gamma|_{j-1}) \cap C[L] = \emptyset, \neg L \in I^p(\Gamma|_{j-1})\}$; and
- $ccgi(A_j \rhd L_j, \Gamma) = \{L : \ C[L] \in ccgi(A_j \rhd C_j[L_j], \Gamma)\}$.

Proper constrained ground instances of $A_j \rhd C_j[L_j]$ are not satisfied in $I^p(\Gamma|_{j-1})$ and are satisfied in $I^p(\Gamma|_j)$, while complementary ones are not satisfied in either interpretation.

Example 3 If Γ is the sequence of unit clauses $P(a,x), P(b,y), \neg P(z,z), P(u,v)$ of Example 2, we have: no ground instance of $P(a,x)$ is complementary; no ground instance of $P(b,y)$ is complementary; the complementary ground instances of $\neg P(z,z)$ are $\neg P(a,a)$ and $\neg P(b,b)$; the complementary ground instances of $P(u,v)$ are those of the form $P(t,t)$ for any ground term t other than a and b.

Proper constrained ground instances enjoy the following *monotonicity* property, where we use juxtaposition to represent concatenation of sequences:

Lemma 1 *For all clauses C and SGGS clause sequences Γ_1, Γ_2, and Γ_3,*

$$pcgi(C, \Gamma_1\Gamma_2 C \Gamma_3) \subseteq pcgi(C, \Gamma_1 C \Gamma_2 \Gamma_3).$$

Proof It follows by construction from Definition 7. □

The meaning of this lemma is that since $I^p(\Gamma)$ is built from left to right, the same clause may contribute more proper constrained ground instances, if it occurs at a smaller index in the sequence. Complementary constrained ground instances do not satisfy this monotonicity property: $ccgi(C,\Gamma_1\Gamma_2C\Gamma_3) \subseteq ccgi(C,\Gamma_1C\Gamma_2\Gamma_3)$ does not hold, because for a $C'[L'] \in Gr(C)$, it can be that $\neg L' \in I^p(\Gamma_1\Gamma_2)$ but $\neg L' \notin I^p(\Gamma_1)$.

If an instance $C[L] \in Gr(A_j \rhd C_j[L_j])$ is not satisfied by $I^p(\Gamma|_{j-1})$, it is either proper ($\neg L \notin I^p(\Gamma|_{j-1})$) or complementary ($\neg L \in I^p(\Gamma|_{j-1})$). Equivalently, if it is neither, it is satisfied by $I^p(\Gamma|_{j-1})$. Thus, if $A_j \rhd C_j[L_j]$ has neither proper nor complementary constrained ground instances, $I^p(\Gamma|_{j-1})$ satisfies all its constrained ground instances, and therefore $A_j \rhd C_j[L_j]$ itself. Such a clause is *disposable*:

Definition 9 (Disposable clause) A non-empty clause $A \rhd C[L]$ in Γ is *disposable*, if $pcgi(A \rhd L,\Gamma) = ccgi(A \rhd L,\Gamma) = \emptyset$.

Indeed, a satisfied clause is not useful to modify $I^p(\Gamma)$. This resembles what happens in DPLL or DPLL-CDCL, when a clause satisfied by the current trail is considered done with. The following example shows how disposability depends on the position of a clause in a sequence, and therefore it differs from subsumption:

Example 4 Consider for simplicity sequences consisting only of unit clauses whose sole literal is selected:

- $\neg P(x),\ \neg Q(x),\ \neg P(x)$: the second $\neg P(x)$ is disposable because it has neither proper nor complementary constrained ground instances;
- $P(x),\ \neg Q(x),\ P(x)$: the second $P(x)$ is disposable for the same reason;
- $\neg P(x),\ \neg Q(x),\ P(x)$: clause $P(x)$ is not disposable because all its instances are complementary constrained ground instances;
- $P(x),\ \neg Q(x),\ \neg P(x)$: the same is true for $\neg P(x)$;
- $P(x),\ \neg P(x),\ \neg P(x)$: neither occurrence of $\neg P(x)$ is disposable, because for both all instances are complementary constrained ground instances;
- $P(z),\ P(f(x)),\ P(a)$: clauses $P(f(x))$ and $P(a)$ are disposable, so that disposability agrees with subsumption;
- $P(a),\ P(f(x)),\ P(z)$: clause $P(z)$ is disposable, if the signature has only one function symbol f and one constant symbol a, so that disposability does not agree with subsumption.

By consulting I whenever $I^p(\Gamma)$ does not determine the truth value of a ground literal, $I^p(\Gamma)$ can be completed in an interpretation:

Definition 10 (Induced interpretation) The *Herbrand interpretation induced* by an SGGS clause sequence Γ, denoted $I[\Gamma]$, is the interpretation such that, for all ground literals L, if $at(L) \in at(I^p(\Gamma))$, then $I[\Gamma] \models L$ if and only if $L \in I^p(\Gamma)$; if $at(L) \notin at(I^p(\Gamma))$, then $I[\Gamma] \models L$ if and only if $I \models L$.

In particular, if $\Gamma = \varepsilon$ then $I[\Gamma] = I$. In other words, $I[\Gamma]$ makes all proper constrained ground instances of all selected literals in Γ true, and otherwise is like I.

Example 5 Let Γ be the sequence of unit clauses $P(a,x), P(b,y), \neg P(z,z), P(u,v)$, of Examples 2 and 3, and I be the all-negative interpretation, where negative literals

are true and positive literals are false. Then $I[\Gamma] \models P(a,t)$ and $I[\Gamma] \models P(b,t)$ for all ground terms t, but $I[\Gamma] \not\models P(t,t)$ for t other than a and b, and $I[\Gamma] \models P(u,v)$ for all distinct ground terms u and v.

Since every prefix of a sequence is a sequence, Definition 10 applies to every prefix, as shown in the following:

Example 6 Let $\Gamma = C_1, C_2, C_3$ be $[P(x)]$, $\neg P(f(y)) \vee [Q(y)]$, $\neg P(f(z)) \vee \neg Q(g(z)) \vee [R(f(z), g(z))]$, again with I all-negative. Then $I[\Gamma|_0] = I[\varepsilon] = I$ interprets all positive literals as false; $I[\Gamma|_1]$ interprets all positive literals as false, except for the ground instances of $P(x)$; $I[\Gamma|_2]$ interprets all positive literals as false, except for the ground instances of $P(x)$ and $Q(y)$; and $I[\Gamma|_3] = I[\Gamma]$ interprets all positive literals as false, except for the ground instances of $P(x)$, $Q(y)$ and $R(f(z), g(z))$.

$I[\Gamma]$ and I differ on the proper constrained ground instances of I-false selected literals, which are true in $I[\Gamma]$ and false in I. On the other hand, as proper constrained ground instances of I-true selected literals are true in I, I-true selected literals do not affect $I[\Gamma]$. Since an I-true literal can be selected only in an I-all-true clause, it follows that I-all-true clauses in Γ do not affect $I[\Gamma]$. The following theorem makes these remarks precise:

Theorem 1 *Let Γ be an SGGS clause sequence $A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$. For all j, $1 \leq j \leq n$, if $A_j \rhd C_j[L_j]$ is not I-all-true, then $I[\Gamma|_j] \neq I[\Gamma|_{j-1}]$ if and only if $pcgi(A_j \rhd C_j[L_j], \Gamma) \neq \emptyset$; if $A_j \rhd C_j[L_j]$ is I-all-true, then $I[\Gamma|_j] = I[\Gamma|_{j-1}]$.*

Proof As $I^p(\Gamma|_j) = I^p(\Gamma|_{j-1}) \cup pcgi(A_j \rhd L_j, \Gamma)$, interpretations $I[\Gamma|_j]$ and $I[\Gamma|_{j-1}]$ may differ only in the interpretation of the literals in $pcgi(A_j \rhd L_j, \Gamma)$. If $A_j \rhd C_j[L_j]$ is not I-all-true, L_j is I-false, by Condition (2) of Definition 5. Then, for all $L \in pcgi(A_j \rhd L_j, \Gamma)$, $I[\Gamma|_j] \models L$, because $L \in I^p(\Gamma|_j)$, but $I[\Gamma|_{j-1}] \not\models L$, because $L \notin I^p(\Gamma|_{j-1})$ by Definition 7 and L is I-false. If $A_j \rhd C_j[L_j]$ is I-all-true, then for all $L \in pcgi(A_j \rhd L_j, \Gamma)$, $I[\Gamma|_j] \models L$, because $L \in I^p(\Gamma|_j)$, and $I[\Gamma|_{j-1}] \models L$, because L is I-true, so that $I[\Gamma|_j] = I[\Gamma|_{j-1}]$. □

Thus, I-all-true clauses do not contribute to build $I[\Gamma]$. What is then their rôle in an SGGS clause sequence? This question sets the stage for the next section, where we study first-order clausal propagation.

5 First-order Clausal Propagation

A major feature of SGGS is a first-order conflict-driven clause learning (CDCL) mechanism. In order to have such a mechanism in the inference system, the model representation approach needs to make available a notion of *propagation*, so that the procedure discovers that a clause is in conflict with the current model by propagation.

Consider DPLL-CDCL applied to a set of propositional clauses: if literal $\neg L$ appears in the trail, all occurrences of L in the set are false; the truth value of L *depends* on the assertion of $\neg L$ in the trail. SGGS generalizes this concept to first-order logic: if all constrained ground instances of a literal L appear negated among the proper constrained ground instances of a selected literal M, literal L is uniformly false in $I[\Gamma]$. The uniform falsity of L *depends* on the selection of M in the sequence:

Definition 11 (Dependence) Given $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$, for all indices j and k, $1 \leq j,k \leq n$, for all literals $L \in C_j[L_j]$, L *depends* on L_k if

1. $k < j$,
2. $\neg Gr(A_j \rhd L) \subseteq pcgi(A_k \rhd L_k, \Gamma)$,
3. $L_k \in flits(C_k[L_k])$ if $L \in tlits(C_j[L_j])$, and
4. $L_k \in tlits(C_k[L_k])$ if $L \in flits(C_j[L_j])$.

Condition (1) is coherent with the fact that $I^p(\Gamma)$, whence $I[\Gamma]$, is built by consulting the sequence from left to right or by increasing indices. Condition (2) ensures that L is uniformly false in $I[\Gamma]$. Conditions (3) and (4) descend from the *semantic guidance* by I: all literals in the sequence are either I-true or I-false, and therefore if all ground instances of one appear negated among ground instances of another, it must be that one is I-true and the other I-false. In other words, whenever there is a dependence, either an I-false selected literal makes uniformly false in $I[\Gamma]$ an I-true literal occurring at a larger index, or an I-true selected literal makes uniformly false in $I[\Gamma]$ an I-false literal occurring at a larger index.

Since $I[\Gamma]$ differs from I on I-false selected literals, the most relevant dependences are those where I-true literals are made uniformly false by I-false selected literals, as in the following:

Example 7 Let Γ be $[P(x)], \neg P(f(y)) \vee [Q(y)], \neg P(f(z)) \vee \neg Q(g(z)) \vee [R(f(z), g(z))]$, with clauses named C_1, C_2, C_3, and all-negative I as in Example 6. The I-true literal $\neg P(f(y))$ in C_2 depends on the selected I-false literal $P(x)$ in C_1, because all ground instances of $P(x)$ are proper, and $P(f(y))$ is an instance of $P(x)$. For the same reason, the I-true literal $\neg P(f(z))$ in C_3 also depends on $P(x)$. Similarly, the I-true literal $\neg Q(g(z))$ in C_3 depends on the selected I-false literal $Q(y)$ in C_2.

In other words, although dependence is define symmetrically, SGGS is interested in the dependences of I-true literals on I-false literals: the semantic guidance by I induces a sort of asymmetry in the method, so that first-order propagation is under semantic guidance by I, or modulo semantic guidance by I. We use *assignment functions*, or *assignments* for short, to represent dependences of I-true literals:

Definition 12 (Assignment) Given $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$, for all $A_j \rhd C_j[L_j]$, $1 \leq j \leq n$, an *assignment* is a partial function $\Phi^j_\Gamma : tlits(C_j) \to \{1, \ldots, n\}$ such that, for all $L \in tlits(C_j[L_j])$, if $\Phi^j_\Gamma(L) = k$, then:

1. L depends on L_k;
2. If $L \neq L_j$, then $\Phi^j_\Gamma(L)$ is defined: I-true literals that are not selected must be assigned;
3. If $L = L_j$ and there exists an L_k such that L depends on L_k, then $\Phi^j_\Gamma(L)$ is defined: selected I-true literals are assigned if possible;
4. If $L \neq L_j$, $L_j \in tlits(C_j)$, and $\Phi^j_\Gamma(L_j) = i$, then $k \leq i < j$: if the selected literal L_j of C_j is I-true and it is assigned, then it is assigned to the largest index (or to the rightmost clause) among all literals of C_j.

By slightly abusing the terminology, we may say that a literal is assigned to a literal or to clause rather than to an index; and we say that a clause *depends* on another clause if a literal of the former is assigned to the latter.

Example 8 Let Γ and I be as in Example 7. The I-true literals in the sequence are $\neg P(f(y))$, $\neg P(f(z))$, and $\neg Q(g(z))$. The assignment function assigns $\neg P(f(y))$ and $\neg P(f(z))$ to $P(x)$; and it assigns $\neg Q(g(z))$ to $Q(y)$. Thus, clauses C_2 and C_3 depend on clause C_1; and clause C_3 also depends on C_2.

An SGGS clause sequence $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$ is implicitly equipped with a set $\Phi_\Gamma = \{\Phi_\Gamma^1, \ldots, \Phi_\Gamma^n\}$ of assignments, one for every clause in Γ. We still need to elucidate Conditions (2), (3) and (4) in Definition 12. These requirements will ensure that I-all-true clauses in Γ are either *in conflict* with $I[\Gamma]$, signaling that Γ must be fixed before being extended further; or are *justifications* of their selected literal as an *implied literal* that must be in $I[\Gamma]$, because there is no other way to satisfy the clause. Before presenting these concepts at the first-order level, we show them at work in an SGGS-refutation of a simple propositional example:

Example 9 Consider the set of clauses $S = \{P \vee Q, P \vee \neg Q, \neg P \vee Q, \neg P \vee \neg Q\}$ with all-negative I, so that $I = \{\neg P, \neg Q\}$. By convention the selected literal is placed rightmost, and clauses in the current sequence are labeled $C_1, C_2, \ldots, C_i, \ldots$. The steps of a derivation are as follows:

1. Clause $P \vee Q$ is not satisfied by I, and therefore an *SGGS-extension* step adds it to $\Gamma_0 = \varepsilon$ producing $\Gamma_1 = Q \vee [P]$. Both P and Q are I-false and either one can be selected: we assume that P is selected, so that $I[\Gamma_1] = \{P, \neg Q\}$ and $I[\Gamma_1] \models P \vee Q$.
2. Clause $\neg P \vee Q$ is not satisfied by $I[\Gamma_1]$, and gets added by *SGGS-extension* yielding $\Gamma_2 = Q \vee [P],\ \neg P \vee [Q]$, where Q is selected in C_2, because it is I-false, while $\neg P$, which is I-true, is assigned to C_1. The induced interpretation $I[\Gamma_2] = \{P, Q\}$ satisfies both C_1 and C_2.
3. Clause $\neg P \vee \neg Q$ is not satisfied by $I[\Gamma_2]$, and *SGGS-extension* extends Γ_2 into $\Gamma_3 = Q \vee [P],\ \neg P \vee [Q],\ \neg P \vee [\neg Q]$. Clause C_3 is I-all-true: its literals $\neg P$ and $\neg Q$ are assigned to C_1 and C_2, respectively, and $\neg Q$ is selected, so that the selected literal is assigned rightmost (cf. Condition (4) in Definition 12). Since it is I-all-true, C_3 does not affect the induced interpretation, $I[\Gamma_3] = I[\Gamma_2] = \{P, Q\}$ (cf. Theorem 1), and C_3 is *in conflict* with $I[\Gamma_3]$. While in the previous steps SGGS-extension adds a clause C_j which is not satisfied by $I[\Gamma|_{j-1}]$, but is satisfied by $I[\Gamma_j]$, when an I-all-true C_j is added, it is in conflict with $I[\Gamma_j]$ itself.
4. It is necessary to fix the sequence and its induced interpretation before further extensions. First, since induced interpretations are built from selected literals, the way to solve the conflict is to modify the sequence so that the selected literal of the I-all-true clause enters the induced interpretation. Second, the selected literal of the I-all-true clause is made false in the current induced interpretation by the literal which it is assigned to (cf. Definition 11). Third, induced interpretations are built from left to right. Thus, SGGS solves the conflict by *moving* the conflicting I-all-true clause C_3 *to the left* of C_2, namely the clause which the selected literal of C_3 is assigned to in Γ_3. The result of this *SGGS-move* inference is $\Gamma_4 = Q \vee [P],\ \neg P \vee [\neg Q], \neg P \vee [Q]$, so that $I[\Gamma_4] = \{P, \neg Q\}$: the move *flips* a literal in the induced interpretation, with an effect similar to that of backjumping in DPLL-CDCL. In Γ_4 the selected I-true literal $\neg Q$ of C_2 is *not* assigned: this signals that $\neg P \vee [\neg Q]$ is no longer a conflict clause; rather, it is the *justification* of why $\neg Q$

is in the model. Furthermore, now we also see the reason for Condition (4) in Definition 12: the requirement that the selected literal of an I-all-true clause, if assigned, be assigned rightmost, ensures that when the I-all-true clause moves to the left of the clause which its selected literal was assigned to, all other literals in the I-all-true clause will still be assigned.

5. Now it is the turn of $\neg P \vee [Q]$, namely C_3 in Γ_4, to be *in conflict* with $I[\Gamma_4]$, but C_3 is *not* I-all-true. Thus, SGGS does not move it, but *explains* the conflict by *resolving* the I-false selected literal in the conflict clause C_3, namely Q, with the implied I-true selected literal $\neg Q$ justified by C_2, since it is the presence of $\neg Q$ in the model that causes the falsity of Q. *SGGS-resolution* resolves C_2 and C_3 in Γ_4 on their selected literals, and *replaces* the parent that is not I-all-true (C_3) by the resolvent $\neg P$, yielding $\Gamma_5 = Q \vee [P],\ \neg P \vee [\neg Q],\ [\neg P]$, where $I[\Gamma_5] = I[\Gamma_4] = \{P, \neg Q\}$, and the I-true selected literal $[\neg P]$ is assigned to C_1.
6. Now the I-all-true clause C_3 in Γ_5 is *in conflict* with $I[\Gamma_5]$: as before, SGGS *moves* it to the left of C_1 generating $\Gamma_6 = [\neg P],\ Q \vee [P],\ \neg P \vee [\neg Q]$, where $I[\Gamma_6] = \{\neg P, \neg Q\}$, $\neg P$ is no longer assigned, C_1 is its justification, and it is no longer in conflict.
7. Now it is the turn of C_2 to be *in conflict* with $I[\Gamma_6]$, and SGGS *explains* the conflict by *SGGS-resolution*, which resolves C_1 and C_2 in Γ_6, and replaces the second parent $Q \vee [P]$ with the resolvent, yielding $\Gamma_7 = [\neg P],\ [Q]$, with $I[\Gamma_7] = \{\neg P, Q\}$. What happened to $\neg P \vee [\neg Q]$? Recall that $\neg P \vee [\neg Q]$ *depended* on $Q \vee [P]$, because when $\neg P \vee [\neg Q]$ was added to the sequence (cf. Step 3), its I-true literal $\neg P$ was assigned to $[P]$. SGGS-resolution *removes* any clause that *depended* on the parent being replaced by the resolvent. Note that $\neg P \vee [\neg Q]$ is satisfied by $I[\Gamma_7]$, and therefore there is no loss of information. This feature of SGGS-resolution did not show in Step 5, because then the parent being removed happened to be the last clause in the sequence.
8. Now clause $P \vee \neg Q$ is not satisfied by $I[\Gamma_7]$, and *SGGS-extension* adds it to produce $\Gamma_8 = [\neg P],\ [Q],\ \neg Q \vee [P]$, where P is selected because it is I-false, the I-true literal $\neg Q$ is assigned to C_2, and $I[\Gamma_8] = \{\neg P, Q\}$.
9. Now C_3 is *in conflict* with $I[\Gamma_8]$, and SGGS *explains* the conflict by resolving the I-false selected literal in the conflict clause C_3, namely P, with the implied I-true selected literal $\neg P$ justified by C_1, since the presence of $\neg P$ in the model causes the falsity of P. The result is $\Gamma_9 = [\neg P],\ [Q],\ [\neg Q]$, as the resolvent replaces the parent which is not I-all-true, and $I[\Gamma_9] = \{\neg P, Q\}$.
10. The I-all-true clause $[\neg Q]$, with its selected literal still assigned to C_2, is in conflict with $I[\Gamma_9]$. Thus, SGGS *moves it left* of C_2, generating $\Gamma_{10} = [\neg P],\ [\neg Q],\ [Q]$, and modifying the induced interpretation to $I[\Gamma_{10}] = \{\neg P, \neg Q\}$.
11. Finally, SGGS resolves $[\neg Q]$ and $[Q]$ in Γ_{10}, yielding $\Gamma_{11} = [\neg P],\ [\neg Q],\ \bot$.

6 Conflict Clauses and Implied Literals

In this section we see how the requirements that all I-true literals that are not selected must be assigned, and selected I-true literals are assigned if possible (cf. Conditions (2) and (3) in Definition 12), determine the rôle of I-all-true clauses in SGGS

clause sequences as either *conflict clauses* or *justifications* of *implied literals*. We begin with a lemma which formalizes the observation that an assigned literal is uniformly false:

Lemma 2 *Let Γ be an SGGS clause sequence $A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$. For all j, $1 \leq j \leq n$, for all literals $L \in tlits(C_j)$, if L is assigned, then L is uniformly false in $I[\Gamma|_{j-1}]$.*

Proof Consider an $L \in tlits(C_j)$ such that $\Phi^j_\Gamma(L) = k$. By Condition (1) of Definition 12, L depends on L_k. By Condition (2) of Definition 11, we have $\neg Gr(A_j \rhd L) \subseteq pcgi(A_k \rhd L_k, \Gamma)$ (*). By Condition (1) of Definition 11, we know that $k < j$, or $k \leq j-1$, whence $pcgi(A_k \rhd L_k, \Gamma) \subseteq I^p(\Gamma|_{j-1})$ by Definition 6. By Definition 10, $I[\Gamma|_{j-1}] \models Q$ for all $Q \in I^p(\Gamma|_{j-1})$ and therefore $I[\Gamma|_{j-1}] \models Q$ for all $Q \in pcgi(A_k \rhd L_k, \Gamma)$. It follows from (*) that $I[\Gamma|_{j-1}]$ uniformly falsifies L. □

By Condition (2) of Definition 12, all literals of an I-all-true clause in Γ that are not selected must be assigned, and therefore by Lemma 2 they are all uniformly false in $I[\Gamma]$. We are left with the selected literal. An I-all-true clause in Γ will be either a conflict clause or a justification depending on whether its selected literal is assigned.

To get there, let us recall that SGGS builds SGGS clause sequences trying to satisfy the input set of clauses S. If $I[\Gamma] \models S$, the search is over. Otherwise, there is a clause $C \in S$ such that $I[\Gamma] \not\models C$. This means that there is a ground instance D of C such that $I[\Gamma] \not\models D$. When $I[\Gamma] \not\models C$ for some $C \in S$, SGGS applies the inference rule *SGGS-extension* to generate from C and Γ a possibly constrained clause $A \rhd E$, such that E is an instance of C, that captures D, in the sense that D is a constrained ground instance of $A \rhd E$. SGGS-extension selects a literal L in E, assigns *all* I-true literals of E according to Definition 12, and *appends* $A \rhd E[L]$ to Γ. The purpose is to add to $I^p(\Gamma)$ the proper constrained ground instances of the selected literal of $A \rhd E[L]$, so that $I[\Gamma]$ satisfies the ground instance of L that appears in D (since D is an instance of E) and therefore D itself.

However, if $E[L]$ is I-all-true, then it is *not* possible to satisfy D by adding the proper constrained ground instances of L, because the following theorem shows that any constrained ground instance D of an I-all-true clause such that $I[\Gamma] \not\models D$ is a complementary constrained ground instance. In other words, the ground instance of L that appears in D already occurs negated in $I^p(\Gamma)$ and therefore cannot be added to fix the interpretation. Indeed, if E is I-all-true, also its ground instance D is I-all-true. Then, D is false in $I[\Gamma]$ due to $I^p(\Gamma)$, not due to I:

Theorem 2 *Let Γ be an SGGS clause sequence $A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$. For all j, $1 \leq j \leq n$, if $A_j \rhd C_j[L_j]$ is I-all-true, then for all $D[L] \in Gr(A_j \rhd C_j[L_j])$, $I[\Gamma|_{j-1}] \not\models D[L]$ if and only if $D[L] \in ccgi(A_j \rhd C_j[L_j], \Gamma)$.*

Proof ($\Rightarrow$) Since $A_j \rhd C_j[L_j]$ is I-all-true, $I \models D[L]$. If $I[\Gamma|_{j-1}] \not\models D[L]$, it means that all literals of $D[L]$, including L, appear negated in $I^p(\Gamma|_{j-1})$. Thus, by Definition 8 we have $D[L] \in ccgi(A_j \rhd C_j[L_j], \Gamma)$.
($\Leftarrow$) Assume that $D[L] \in ccgi(A_j \rhd C_j[L_j], \Gamma)$. By Definition 8, no literal of $D[L]$ is in $I^p(\Gamma|_{j-1})$ and $\neg L \in I^p(\Gamma|_{j-1})$. By Condition (2) of Definition 12, all literals

of $A_j \rhd C_j[L_j]$ other than L_j are assigned, and therefore by Lemma 2 they are all uniformly false in $I[\Gamma|_{j-1}]$. It follows that all literals in $D[L]$ other than L are false in $I[\Gamma|_{j-1}]$. Since for L we already established that $\neg L \in I^p(\Gamma|_{j-1})$, we have that $I[\Gamma|_{j-1}] \not\models D[L]$. □

Thus, D is false in $I[\Gamma]$ because of how $I[\Gamma]$ was made different from I in an attempt to satisfy other clauses. For instances at the propositional level see Steps 3, 6, and 10 in Example 9. A simple first-order example is the following:

Example 10 Let Γ be $[P(x)],\ [Q(y)],\ \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)]$ with all-negative I. $I[\Gamma]$ makes all ground instances of $P(x)$, $Q(y)$, and $R(x)$ true, and all other positive ground literals false. Assume that the input set of clauses S contains $\neg R(c)$. SGGS-extension detects that $\neg R(c)$ and the I-false selected literal $R(x)$ in the third clause unify and have opposite sign; thus, it extends Γ to $[P(x)],\ [Q(y)],\ \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$. The last clause $[\neg R(c)]$ is I-all-true and its single literal is assigned to the I-false selected literal $[R(x)]$ in the third clause. Clause $[\neg R(c)]$ is in conflict with $I[\Gamma]$, due to the prior decision of satisfying clause $\neg P(f(x)) \vee \neg Q(g(x)) \vee R(x)$ by making $R(x)$ true.

Let us see how SGGS-extension operates. SGGS-extension applies if there are clauses $C \in S$, and $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$ in Γ, such that the selected literals $M_1, \ldots, M_n$ are I-false, and for distinct literals $L_1, \ldots, L_n$ of C there is a simultaneous most general unifier α such that $L_j\alpha = \neg M_j\alpha$, for all j, $1 \leq j \leq n$. Then SGGS-extension adds an instance of the clause $(\bigwedge_{j=1}^n B_j\alpha) \rhd C\alpha$, namely $A \rhd E[L] = (\bigwedge_{j=1}^n B_j\alpha\beta) \rhd C\alpha\beta$. Since $M_1, \ldots, M_n$ are I-false, and $L_j\alpha = \neg M_j\alpha$, $1 \leq j \leq n$, it follows that $L_1\alpha, \ldots, L_n\alpha$ are I-true, and therefore $L_1\alpha\beta, \ldots, L_n\alpha\beta$ are I-true. The application of the second substitution β guarantees that all other literals of E are I-false, so that $L_1\alpha\beta, \ldots, L_n\alpha\beta$ are *all* the I-true literals of the newly added clause. Furthermore, SGGS-extension assigns them to the clauses $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$. Thus, SGGS-extension guarantees that *all* I-true literals of a newly generated clause are assigned, as required by Definition 12. When the newly added clause is I-all-true, *all* its literals are assigned. Such a clause is *a conflict clause*:

Theorem 3 *Let Γ be an SGGS clause sequence $A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$. For all j, $1 \leq j \leq n$, if $A_j \rhd C_j[L_j]$ is I-all-true, and all literals in $tlits(C_j)$ are assigned, then $A_j \rhd C_j[L_j]$ is uniformly false in $I[\Gamma|_{j-1}]$ and in $I[\Gamma|_j]$.*

Proof By Lemma 2, all literals in $tlits(C_j)$ are uniformly false in $I[\Gamma|_{j-1}]$. Since C_j is I-all-true, this means that all its literals are uniformly false in $I[\Gamma|_{j-1}]$. It follows that C_j is uniformly false in $I[\Gamma|_{j-1}]$ and therefore in $I[\Gamma|_j]$ by Theorem 1. □

The following corollary of Theorems 2 and 3 summarizes all the facts about I-all-true clauses whose literals are all assigned:

Corollary 1 *Let Γ be an SGGS clause sequence $A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$. For all j, $1 \leq j \leq n$, if $A_j \rhd C_j[L_j]$ is I-all-true, and all its literals are assigned, then $Gr(A_j \rhd C_j[L_j]) = ccgi(A_j \rhd C_j[L_j], \Gamma)$, $pcgi(A_j \rhd C_j[L_j], \Gamma) = \emptyset$, and $A_j \rhd C_j[L_j]$ is not disposable.*

Proof By Theorem 3, $I[\Gamma|_{j-1}] \not\models D[L]$ for all $D[L] \in Gr(A_j \rhd C_j[L_j])$. It follows by Theorem 2 that $Gr(A_j \rhd C_j[L_j]) = ccgi(A_j \rhd C_j[L_j], \Gamma)$. Then $pcgi(A_j \rhd C_j[L_j], \Gamma) = \emptyset$, and $A_j \rhd C_j[L_j]$ is not disposable because $ccgi(A_j \rhd C_j[L_j], \Gamma) \neq \emptyset$. □

Obviously, a conflict clause is *not* disposable: a disposable clause is already satisfied and therefore can be ignored; a conflict clause signals that there is a problem, because the interpretation built so far makes it uniformly false.

We consider next an I-all-true clause $A_j \rhd C_j[L_j]$ in Γ where the selected literal L_j is *not* assigned: by Lemma 2, all literals of C_j except L_j are uniformly false in $I[\Gamma|_{j-1}]$. In DPLL-CDCL, if there is a propositional clause C where only one literal L is undefined and all other literals are false with respect to the trail, L is an *implied literal*, because it is implied by C and the literals in the trail, and making it true is the only way to satisfy C. When such a situation is detected, the implied literal is added to the trail with its clause as *justification*. Analogously, L_j is an *implied literal*. Indeed, by Conditions (1) of Definition 12 and (2) of Definition 11, for all $L \in tlits(C_j) \setminus \{L_j\}$, we have $\neg Gr(A_j \rhd L) \subseteq pcgi(A_k \rhd L_k)$, if $\Phi^j_\Gamma(L) = k$, so that $A_k \rhd L_k$ implies $A_j \rhd \neg L$. It follows that the set of clauses $\{A_j \rhd C_j[L_j]\} \cup \{A_k \rhd L_k : \ \Phi^j_\Gamma(L) = k,\ L \in tlits(C_j) \setminus \{L_j\}\}$ implies $A_j \rhd L_j$. In DPLL-CDCL an implied literal depends on the decision levels where all other literals in its clause are made false by either decisions or propagations. Similarly, in SGGS L_j depends on the indices which the other literals in $A_j \rhd C_j[L_j]$ are assigned to.

SGGS-extension guarantees that when an I-all-true clause is added, all its literals are assigned. All other SGGS inference rules preserve assignments. How can then Γ contains an I-all-true $A_j \rhd C_j[L_j]$ such that L_j is not assigned? As an effect of an *SGGS-move* step, as we saw in Steps 4, 6 and 10 of Example 9. A simple first-order example follows:

Example 11 Let Γ be $[P(x)],\ [Q(y)],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ \neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)],\ [\neg R(c)]$ with I all-negative. $I[\Gamma]$ makes all ground instances of $P(x)$, $Q(y)$, and $R(x)$ true, and all other positive ground literals false. The last clause $[\neg R(c)]$ is I-all-true, it is in conflict with $I[\Gamma]$, and its single literal is assigned to the I-false selected literal $[R(c)]$ in the fourth clause. An SGGS-move of $[\neg R(c)]$ to the left of the fourth clause yields $[P(x)],\ [Q(y)],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)],\ \neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)]$, and causes the truth value of $\neg R(c)$ to flip from *false* to *true* in $I[\Gamma]$.

We illustrate this mechanism in general. When SGGS-extension appends to Γ an I-all-true clause $A \rhd E[L]$, all literals in $E[L]$ are assigned, and $A \rhd E[L]$ is a conflict clause. Let $B \rhd D[M]$ be the clause which L is assigned to. SGGS *solves the conflict* by moving $A \rhd E[L]$ to the left of $B \rhd D[M]$, so that L is no longer assigned. By Lemma 2, L was made uniformly false by M. Since all constrained ground instances of L appeared negated among the proper constrained ground instances of M (cf. Condition 2 in Definition 11) before the move, *all* constrained ground instances of L are *proper* after the move. The effect of the move is to *flip* at once the truth value of all constrained ground instances of L in $I[\Gamma]$ from *false* to *true*. The I-all-true clause $A_j \rhd C_j[L_j]$ whose selected literal L_j is not assigned stands in Γ as the *justification* of L_j. Such

a clause is obviously not disposable, as $Gr(A_j \rhd C_j[L_j]) = pcgi(A_j \rhd C_j[L_j], \Gamma)$ and $ccgi(A_j \rhd C_j[L_j], \Gamma) = \emptyset$.

By requiring that all I-true literals that are not selected are assigned, SGGS ensures that I-all-true clauses in Γ are either conflict clauses or justifications of implied literals. This clarifies the meaning of Conditions (2) and (3) in Definition 12. Condition (4) is related to the CDCL mechanism of SGGS as we saw in Step 4 of Example 9: the selected literal of an I-all-true clause is assigned rightmost so that it is the only one which is no longer assigned when the clause moves left.

Two more remarks are in order. First, by Theorem 1 we know that selected I-true literals affect $I^p(\Gamma)$, not $I[\Gamma]$. In other words, implied literals from I-all-true clauses affect $I^p(\Gamma)$, not $I[\Gamma]$. Then, do these literals have an operational rôle? Second, also a clause which is not I-all-true can be a conflict clause: this happens if all its literals are uniformly false in $I[\Gamma]$. Then, the operational rôle of implied literals from I-all-true clauses is to resolve away the I-false literals in conflict clauses that are not I-all-true. We saw instances of this mechanism in Steps 5, 7, and 9 of Example 9. Once all I-false literals in the conflict clause have been resolved away, we have either an empty clause or an I-all-true conflict clause. In the first case a refutation has been found. In the second case the I-all-true conflict clause is subject to an SGGS-move as described above. The series of SGGS-resolution steps that resolves away the I-false literals in the conflict clause *explains* the conflict like in DPLL-CDCL. Both DPLL-CDCL and SGGS explain conflicts by resolving literals in the conflict clause with implied literals in the trail or in the sequence, respectively. The differences are that DPLL-CDCL uses propositional resolution, while SGGS-resolution is first-order, and SGGS is semantically guided by I, so that it is I-false literals that get resolved away. This discussion leads us to the next section, where we analyze those clauses in the sequence that are not I-all-true.

7 The Logical Meaning of SGGS Clause Sequences

In this section we complete the analysis by considering the clauses that are *not* I-all-true in an SGGS clause sequence, and we conclude with a theorem on how from a sequence Γ follows a formula capturing both $I[\Gamma]$ and other interpretations not yet considered. We begin with the companion of Theorem 2 for clauses that are not I-all-true: it shows that the constrained ground instances of $A_j \rhd C_j[L_j]$ that are not satisfied by $I[\Gamma|_{j-1}]$ are its proper or complementary constrained ground instances:

Theorem 4 *Let Γ be an SGGS clause sequence $A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$. For all j, $1 \leq j \leq n$, if $A_j \rhd C_j[L_j]$ is not I-all-true, then for all $D[L] \in Gr(A_j \rhd C_j[L_j])$, $I[\Gamma|_{j-1}] \not\models D[L]$ if and only if $D[L] \in pcgi(A_j \rhd C_j[L_j], \Gamma) \cup ccgi(A_j \rhd C_j[L_j], \Gamma)$.*

Proof ($\Rightarrow$) If $I[\Gamma|_{j-1}] \not\models D[L]$, then no literal of $D[L]$ appears in $I^p(\Gamma|_{j-1})$, so that $I^p(\Gamma|_{j-1}) \cap D = \emptyset$. If $\neg L \notin I^p(\Gamma|_{j-1})$ then $D[L] \in pcgi(A_j \rhd C_j[L_j], \Gamma)$. If $\neg L \in I^p(\Gamma|_{j-1})$ then $D[L] \in ccgi(A_j \rhd C_j[L_j], \Gamma)$.
($\Leftarrow$) If $D[L] \in pcgi(A_j \rhd C_j[L_j], \Gamma) \cup ccgi(A_j \rhd C_j[L_j], \Gamma)$, then by Definitions 7 and 8 we have $I^p(\Gamma|_{j-1}) \cap D = \emptyset$. For all literals M of $D[L]$, there are two cases:

1. $M \in flits(D[L])$: if $\neg M$ is in $I^p(\Gamma|_{j-1})$, then $I[\Gamma|_{j-1}] \not\models M$; otherwise, $I[\Gamma|_{j-1}] \not\models M$ because $I \not\models M$.
2. $M \in tlits(D[L])$: let Q be the I-true literal of $A_j \rhd C_j[L_j]$ such that M is an instance of Q. Since C_j is not I-all-true, by Condition (2) of Definition 5, Q is not selected. By Condition (2) of Definition 12, Q is assigned, and by Lemma 2, it is uniformly false in $I[\Gamma|_{j-1}]$, so that $I[\Gamma|_{j-1}] \not\models M$.

Since for all literals M of $D[L]$ we have $I[\Gamma|_{j-1}] \not\models M$, it follows that $I[\Gamma|_{j-1}] \not\models D[L]$. □

As a straightforward corollary of this theorem, a clause satisfied by the interpretation induced by the clauses on its left in the sequence is *disposable*:

Corollary 2 *Let Γ be an SGGS clause sequence $A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$. For all j, $1 \leq j \leq n$, if $A_j \rhd C_j[L_j]$ is not I-all-true, $I[\Gamma|_{j-1}] \models A_j \rhd C_j[L_j]$ if and only if $A_j \rhd C_j[L_j]$ is disposable.*

A clause that is not I-all-true and is not disposable can be in conflict. The next theorem characterizes this situation. In order to state it we need first the following:

Definition 13 (Intersection) Constrained literals $A \rhd L$ and $B \rhd M$ have non-empty *intersection*, or *intersect*, if $at(Gr(A \rhd L)) \cap at(Gr(B \rhd M)) \neq \emptyset$.

Note that intersection is defined based on atoms, that is, regardless of sign.

Theorem 5 *Let Γ be an SGGS clause sequence $A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$. For all j, $1 \leq j \leq n$, if $A_j \rhd C_j[L_j]$ is not I-all-true, and no element of $flits(C_j)$ intersects any I-false L_i for $i < j$, then $A_j \rhd C_j[L_j]$ is uniformly false in $I[\Gamma|_{j-1}]$.*

Proof By Condition (1) of Definition 5, every literal L of C_j is either I-false or I-true. Thus, there are two cases:

1. $L \in tlits(C_j)$: since C_j is not I-all-true, by Condition (2) of Definition 5, L is not selected. By Condition (2) of Definition 12, L is assigned, and by Lemma 2, it is uniformly false in $I[\Gamma|_{j-1}]$.
2. $L \in flits(C_j)$: for no $L' \in Gr(L)$ it can be that $L' \in I^p(\Gamma|_{j-1})$, because, by hypothesis, L does not intersect any I-false L_i with $i < j$, or, equivalently, $i \leq j-1$. (Note that intersection with an I-true L_i with $i < j$ is not relevant here, because it would mean $\neg L' \in I^p(\Gamma|_{j-1})$.) Thus, by Definition 10, L is uniformly false in $I[\Gamma|_{j-1}]$.

Since all literals of $A_j \rhd C_j[L_j]$ are uniformly false in $I[\Gamma|_{j-1}]$, the claim holds. □

The above theorem gives sufficient conditions for a non-I-all-true clause to be a conflict clause. In the context of SGGS-extension, stronger conditions may be satisfied. Recall (cf. Section 6) how SGGS-extension applies if there are clauses $C \in S$, and $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$ in Γ, such that the selected literals $M_1, \ldots, M_n$ are I-false, and for distinct literals $L_1, \ldots, L_n$ of C there is a simultaneous most general unifier α such that $L_j\alpha = \neg M_j\alpha$, for all j, $1 \leq j \leq n$. Then SGGS-extension generates $A \rhd E[L] = (\bigwedge_{j=1}^{n} B_j\alpha\beta) \rhd C\alpha\beta$. If E is not I-all-true and every I-false literal

$L \in flits(E)$ intersects some I-true selected literal L_i in Γ, SGGS-extension *appends* to Γ the instance $A\gamma \rhd E[L]\gamma$, where γ is the simultaneous most general unifier of the I-false literals of E with those intersecting I-true selected literals. This substitution γ is called *extension substitution*. It follows that not only the I-false literals of the appended clause do not intersect I-false selected literals of smaller index, but they are subsumed by I-true selected literals of smaller index. This ensures that if SGGS-extension appends to Γ a conflict clause that is not I-all-true, it will be possible to resolve away all its I-false literals by resolving the conflict clause with the justifications of those subsuming I-true selected literals of smaller index. In other words, it will be possible to *explain* the conflict.

Having dealt with the situations where SGGS-extension appends a conflict clause, we are left with the case where SGGS-extension adds a clause that contributes to build $I[\Gamma]$. The following theorem shows that sufficient conditions for this to happen is to have a clause with I-false selected literal that does not intersect I-true selected literals of smaller index:

Theorem 6 *Let Γ be an SGGS clause sequence $A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$. For all j, $1 \leq j \leq n$, if L_j is I-false and does not intersect any I-true L_i for $i < j$, then $I[\Gamma|_j] \models A_j \rhd C_j[L_j]$.*

Proof In order to establish that $I[\Gamma|_j] \models A_j \rhd C_j[L_j]$, we need to show that $I[\Gamma|_j] \models C'$ for all $C' \in Gr(A_j \rhd C_j[L_j])$. For any such C' there are two cases: either $C' \cap I^p(\Gamma|_{j-1}) \neq \emptyset$ or $C' \cap I^p(\Gamma|_{j-1}) = \emptyset$. We distinguish these two cases:

1. If $C' \cap I^p(\Gamma|_{j-1}) \neq \emptyset$, then $C' \cap I^p(\Gamma|_j) \neq \emptyset$, and $I[\Gamma|_j] \models C'$;
2. If $C' \cap I^p(\Gamma|_{j-1}) = \emptyset$, let L' be the instance of L_j in C'. The hypothesis that L_j does not intersect any I-true selected literals of smaller index implies that $\neg L' \notin I^p(\Gamma|_{j-1})$. Thus, by Definition 7, C' and L' are proper constrained ground instances, and $L' \in I^p(\Gamma|_j)$. Therefore, $I[\Gamma|_j] \models L'$ by Definition 10, whence $I[\Gamma|_j] \models C'$.

Since for all $C' \in Gr(A_j \rhd C_j[L_j])$, it holds that $I[\Gamma|_j] \models C'$, we have that $I[\Gamma|_j] \models A_j \rhd C_j[L_j]$. □

In other words, $A_j \rhd C_j[L_j]$ is satisfied by $I[\Gamma|_j]$, because all its constrained ground instances are satisfied: some may be satisfied by $I[\Gamma|_{j-1}]$, or the interpretation induced by clauses of smaller index; all those that are not satisfied by $I[\Gamma|_{j-1}]$ are satisfied by adding to $I^p(\Gamma|_{j-1})$ the proper constrained ground instances of $A_j \rhd L_j$ to form $I^p(\Gamma|_j)$.

Example 12 We revisit Example 6 generating its Γ by SGGS-extensions that obey Theorem 6. We start with $\Gamma_0 = \varepsilon$, I all-negative, and clauses $C_1 = P(x)$, $C_2 = \neg P(f(y)) \vee Q(y)$, and $C_3 = \neg P(f(z)) \vee \neg Q(g(z)) \vee R(f(z), g(z))$ in S. Clause C_1 is not satisfied by I, and trivially does not intersect any previous selected literal in the sequence. SGGS-extension generates $\Gamma_1 = [P(x)]$ and $I[\Gamma_1] \models C_1$. Clause C_2 is not satisfied by $I[\Gamma_1]$, and has an I-false literal, namely $Q(y)$, which does not intersect the selected literal $P(x)$ in Γ_1. SGGS-extension generates $\Gamma_2 = [P(x)],\ \neg P(f(y)) \vee [Q(y)]$ and $I[\Gamma_2] \models C_1,\ C_2$. Clause C_3 is not satisfied by $I[\Gamma_2]$, and has an I-false literal, namely

$R(f(z),g(z))$, which does not intersect the selected literals $P(x)$ and $Q(y)$ in Γ_2. SGGS-extension generates $\Gamma_3 = [P(x)],\ \neg P(f(y)) \vee [Q(y)],\ \neg P(f(z)) \vee \neg Q(g(z)) \vee [R(f(z),g(z))]$ and $I[\Gamma_3] \models C_1,\ C_2,\ C_3$.

Theorem 6 cooperates with Theorem 5 to explain why SGGS-extension applies the extension substitution. Let $A \rhd E[L]$ be, as above, the clause generated by SGGS-extension prior to applying the extension substitution. If E is not I-all-true, and every I-false literal $L \in flits(E)$ intersects some I-true selected literal L_i in Γ, it means that it is not possible to fulfill the condition of Theorem 6 and extend the induced interpretation to satisfy the new clause. More precisely, there is no I-false literal in E that can be selected to satisfy $A \rhd E[L]$. Then, since it is not possible to satisfy $A \rhd E[L]$, SGGS-extension turns it into a conflict clause by applying the extension substitution. This is coherent with the whole approach of SGGS of generalizing to the first-order level the propagation and conflict solving features of DPLL-CDCL: in propositional logic, if something is not true, then it is false; in first-order logic, if something is not true, it only has a false ground instance; then SGGS instantiates it further so that it is uniformly false, and can be dealt with as a conflict.

The last theorem views an SGGS clause sequence as the conjunction of its clauses, and shows that this conjunction entails a formula, which captures the current induced interpretation as well as alternative interpretations:

Theorem 7 *If Γ is an SGGS clause sequence $A_1 \rhd C_1[L_1],\ldots,A_n \rhd C_n[L_n]$, let Γ stand for the conjunction of its clauses, and let $\exists(F)$ be the existential closure of the disjunction of all I-false constrained literals in $\{A_1 \rhd C_1[L_1],\ldots,A_n \rhd C_n[L_n]\}$ that are not selected in Γ. Then $\Gamma \models \exists(F) \vee [(A_1 \rhd L_1) \wedge \ldots \wedge (A_n \rhd L_n)]$.*

Proof Assume that J is a model of Γ seen as the conjunction of its clauses. If $J \models A_1 \rhd L_1 \wedge \ldots \wedge A_n \rhd L_n$ the conclusion follows. Otherwise, let i be the smallest index such that $J \not\models A_i \rhd L_i$. Then $J \models A_j \rhd L_j$ for all $j < i$. Since $J \not\models A_i \rhd L_i$, we consider the literals of $A_i \rhd C_i[L_i]$ that are not selected. We consider first I-true literals that are not selected. Let Q be any literal in $tlits(C_i)$ that is not selected. By Condition (2) in Definition 12, Q is assigned. By Condition (1) in Definition 12 and Condition (1) in Definition 11, Q is assigned to a j such that $j < i$. By Condition (2) of Definition 11, we have $\neg Gr(A_i \rhd Q) \subseteq pcgi(A_j \rhd L_j, \Gamma)$: from this and $J \models A_j \rhd L_j$ for all $j < i$, it follows that $J \not\models Q$. Since $J \models \Gamma$, we have $J \models A_i \rhd C_i$, that is, $J \models Gr(A_i \rhd C_i)$. Since $J \not\models A_i \rhd L_i$ and $J \not\models Q$ for every $Q \in tlits(C_i)$ that is not selected, it means that there are constrained ground instances $C' \in Gr(A_i \rhd C_i)$, such that J satisfies C' by satisfying an instance of a non-selected I-false literal in $A_i \rhd C_i$, or an instance of a literal in F. Thus, J satisfies $\exists(F)$, and the result follows. □

The conjunction of the selected literals refers to the current partial interpretation, while the existential closure of the disjunction of non-selected I-false literals refers to other possible interpretations.

8 Comparison with Related Work

Structures to survey interpretations in the literature often have a tree shape, as in *semantic trees* (e.g., [24,46]) and *tableaux* (e.g., [13,63,8,65,64]), where each branch

represents an interpretation. For example, DPLL, FDPLL [4], and the model evolution calculus (MEC) [10,5,6,11,9] use semantic trees, which branch out over complementary literals, while tableaux are trees that branch out over the literals of clauses. The original formulation of model elimination (ME) used *chains* of literals [68], distinguishing between A-literals, inserted in the candidate model, and B-literals, still to be satisfied. It is known since [7] that an ME-chain corresponds to an ME-tableau, where closed branches are omitted, A-literals label inner nodes on a branch representing the current candidate model, and B-literals label frontier nodes.

Once (refutational) completeness is secured, sought-after features of theorem-proving method include *semantic guidance*, *goal sensitivity*, and *proof confluence*. A theorem-proving method is deemed *semantically guided*, if its operations are guided by a given interpretation. This notion originated from *semantic resolution* [84], which included *hyperresolution* [81], and resolution with set of support as special cases. The guiding interpretation remains *fixed*, so that semantic guidance is a different concept from being model-based in the sense of building and evolving a candidate model. SGGS is semantically guided by the initial interpretation I.

A theorem-proving method is deemed *goal sensitive*, if it generates only clauses connected with the goal to be refuted. This notion originated from *resolution with set of support* [94]. The *given clause algorithm*, which is at the heart of resolution and superposition based provers, was originally conceived to implement resolution with set of support, so that this strategy remains available in contemporary provers (e.g., [93,57,82]). Tableaux-based methods aim at goal sensitivity by starting the tableau with a goal clause. Model-driven instance-based methods aim at goal sensitivity by picking as initial candidate model one that satisfies the clauses of S issued from the assumptions, not those issued from the theorem. The latter form the *input set of support* (*iSOS*). SGGS is goal-sensitive under the same assumption for the guiding interpretation I. If there is no theorem (e.g., in consistency checking), one may take as *iSOS* the complement of the largest consistent subset of the input set S. Semantic guidance and goal sensitivity are regarded as useful, if the problem features many axioms or a large knowledge base.

A theorem-proving method is deemed *proof confluent*, if committing to an inference step does not prevent it from finding a proof, so that backtracking is not needed. For example, resolution is proof confluent, because generating a resolvent cannot preclude a proof. ME-tableaux are not proof confluent, because an instantiation of the variables of the tableaux may preclude closing it. Methods that combine instance generation and tableaux, such as the *disconnection calculus* [13,64,66,67], and *hypertableaux* [3,12], are proof confluent, because they add instances to the set of clauses, rather than instantiate the variables in the tableau. Instance-based methods such as FDPLL [4], the model evolution calculus (MEC) [10,5,6,11,9], and Inst-Gen [42,43,55,54] are also proof confluent. To see this, consider DPLL: finding a proof means closing, by finding a contradiction, every branch of a semantic tree. The state of a derivation is given by the semantic tree, every decision or propagation helps to close it eventually, and DPLL is proof confluent. In practice, DPLL uses *depth-first search*, keeps in memory only a branch of the tree (the *context*) at a time, and uses *backtracking* to go from a branch to the next. The same considerations apply to FD-

PLL and MEC, with a first-order semantic tree, depth-first search with backtracking, and *iterative deepening* on term depth to preserve fairness.

This discussion suggests that confluence depends on how the state and the goal of the search are defined. For example, if we view DPLL as a procedure searching for a model, rather than a proof, the goal is no longer to close the semantic tree, but rather to find a branch where all clauses are satisfied. In this view, a state is a branch, not a tree, and the procedure is not confluent, since assigning *true* to L prevents it from succeeding, if no model of the set of clauses contains L. Thus, we could conclude that DPLL is proof confluent, but *not model confluent*. However, model confluence may not be meaningful, since a procedure searching for a model has to make guesses. First-order procedures are seen primarily as searching for a proof, since model building is not semi-decidable in first-order logic.

SGGS is *proof confluent*, with the state of the search given by an SGGS clause sequence. At each step the new clause sequence replaces the old one, so that only one clause sequence exists at any time. It is not necessary to undo previous steps; rather, the current induced interpretation is updated by *resolution* steps between clauses in the sequence, and by *moving* a clause to a smaller index, because the induced interpretation is built by consulting the sequence from left to right. This move has an effect similar to a *backjumping* in DPLL-CDCL, but without going back to a previous state. Since an SGGS clause sequence encodes also other interpretations, in addition to the current one (cf. Theorem 7), proof confluence with respect to SGGS clause sequences may be regarded as similar to proof confluence with respect to semantic trees. However, the SGGS clause sequence, unlike the semantic tree, is the structure that the procedure actually keeps in memory and manipulates: it is a generalization to the first-order level of the *trail* of literals of DPLL-CDCL.

SGGS has several features in common with other methods. It inherits semantic guidance from semantic resolution, and goal-sensitivity from the set-of-support strategy. It shares with instance-based methods the notion of instance generation as main inference mechanism, and with hyperresolution, hypertableaux, and hyperlinking the concept of hyperinference (SGGS-extension). Like MEC it is a model-based first-order method. SGGS is also different from all previous methods: for instance, it differs from MEC in the approach to model representation and the usage of I. MEC generalizes the splitting of DPLL, whereas SGGS generalizes propagation, explanation, and conflict solution of DPLL-CDCL, so that there is no splitting à la DPLL/MEC in SGGS. In general, SGGS differs from all previous instance-based methods, because it does not feature either fine-grained or coarse-grained interleaving of instance generation and propositional, or ground, reasoning.

9 Discussion

Semantically-Guided Goal-Sensitive reasoning (SGGS) lifts DPLL-CDCL to first-order logic, realizing a first-order theorem-proving method that is simultaneously *model-based*, *semantically-guided*, *proof-confluent*, flexible with respect to *goal-sensitivity*, and capable of *conflict-driven model repair*. In this article, we presented the

SGGS approach to model representation by *SGGS clause sequences*, and a series of results connecting it with basic mechanisms of the SGGS inference system.

SGGS clause sequences are made of *constrained clauses* with *selected literals*. Every literal in the sequence is required to be either *true* (I-true) or *uniformly false* (I-false) in a given initial interpretation I, so that it states the truth or falsity in I of *all* its (constrained) ground instances. Since I-true literals are not useful to build an interpretation different from I, I-false literals are preferred for selection, with the exception of I-all-true clauses, namely clauses whose literals are all I-true.

An SGGS clause sequence Γ induces a partial interpretation $I^p(\Gamma)$ built by having each selected literal contribute its ground instances not already supplied by selected literals of smaller index. Ground instances that are complementary to literals already in $I^p(\Gamma)$ are not added, so that $I^p(\Gamma)$ is consistent. $I^p(\Gamma)$ is completed into an interpretation $I[\Gamma]$ by consulting I for those literals whose truth value is not determined by $I^p(\Gamma)$. Thus, $I[\Gamma]$ differs from I in the interpretation of I-false selected literals.

If all constrained ground instances of an I-true literal L appear among the ground instances that an I-false selected literal M of smaller index contributes to $I^p(\Gamma)$, it follows that L is uniformly false in $I[\Gamma]$ (cf. Lemma 2). SGGS uses an *assignment function* to represent this sort of *first-order propagation*, and it requires that all I-true literals that are not selected be assigned. This ensures that an I-all-true clause in Γ is either a *conflict clause* (cf. Theorem 3), or the *justification* of its selected literal. In the latter case, selected literal and I-all-true clause correspond to *implied literal* and *justification* in DPLL-CDCL.

SGGS builds SGGS clause sequences by *SGGS-extension*, an inference rule which adds to the sequence instances of input clauses, in order to satisfy them (cf. Theorem 6). If SGGS-extension produces a conflict clause (cf. Theorems 3 and 5), SGGS first resolves away all the I-false literals of the conflict clause, by resolving it with justifications of implied literals in Γ. This *explanation by first-order resolution*, corresponds to explanation by propositional resolution in DPLL-CDCL. If the explanation process produces the empty clause, the set of clauses is unsatisfiable. Otherwise, the explanation process produces a clause which is still in conflict with $I[\Gamma]$, but is I-all-true. This clause corresponds to an *asserting clause* in DPLL-CDCL. SGGS *moves* it to a position in Γ such that its selected literal becomes *implied*, and therefore enters the model, so that the conflict is solved by amending Γ and $I[\Gamma]$. This move has the effects that *backjumping* has in DPLL-CDCL. The last technical result of this paper (cf. Theorem 7) shows how an SGGS clause sequence represents a snapshot of the model search process, because it entails a formula that portrays the current induced interpretation and a realm of other interpretations yet to be traversed.

The inception of a new approach to model-based reasoning opens many directions for future research. Major issues are to devise efficient mechanisms to implement first-order clausal propagation, and to provide non-trivial (i.e., not based on sign) initial interpretations, as done for instance for OSHL at the ground level in [77]. A different approach could be to make I *dynamic*, as a parameter that can be varied heuristically during the derivation: heuristics in DPLL-based SAT-solvers might be a source of inspiration. Relations between SGGS clause sequences and other formalisms to represent models, and the complexity of SGGS operations, such as constraint standardization, may also be investigated. The study of SGGS as a model-

building method, or as a basis for decision procedures for decidable fragments, and its extension to equality and theory reasoning, are additional subjects for the long term. Theorem proving has made giant strides, and existing systems are very sophisticated (e.g., [5,93,55,57,82]). The challenge of SGGS is to go towards a semantically-oriented style of reasoning, that might pay off for hard problems or new domains.

References

1. L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.
2. 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.
3. Peter Baumgartner. Hyper tableaux - the next generation. In Harrie de Swart, editor, *Proceedings of the 7th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 60–76. Springer, 1998.
4. Peter Baumgartner. FDPLL - A first-order Davis-Putnam-Logeman-Loveland procedure. In David McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 200–219. Springer, 2000.
5. Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the model evolution calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.
6. Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Lemma learning in the model evolution calculus. In Miki Hermann and Andrei Voronkov, editors, *Proceedings of the 13th Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 572–586. Springer, 2006.
7. Peter Baumgartner and Ulrich Furbach. Consolution as a framework for comparing calculi. *Journal of Symbolic Computation*, 16(5):445–477, 1993.
8. Peter Baumgartner and Ulrich Furbach. Variants of clausal tableaux. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction - A Basis for Applications*, volume I: Foundations - Calculi and Methods, chapter 3, pages 73–102. Kluwer Academic Publishers, 1998.
9. Peter Baumgartner, Björn Pelzer, and Cesare Tinelli. Model evolution calculus with equality - revised and implemented. *Journal of Symbolic Computation*, 47(9):1011–1045, 2012.
10. Peter Baumgartner and Cesare Tinelli. The model evolution calculus as a first-order DPLL method. *Artificial Intelligence*, 172(4/5):591–632, 2008.
11. Peter Baumgartner and Uwe Waldmann. Superposition and model evolution combined. In Renate Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 17–34. Springer, 2009.
12. 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 the 24th International Conference on Automated Deduction (CADE)*, volume 7898 of *Lecture Notes in Artificial Intelligence*, pages 126–134. Springer, 2013.
13. Jean-Paul Billon. The disconnection method. In Pierangelo Miglioli, Ugo Moscato, Daniele Mundici, and Mario Ornaghi, editors, *Proceedings of the 5th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, volume 1071 of *Lecture Notes in Artificial Intelligence*, pages 110–126. Springer, 1996.
14. 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.
15. Maria Paola Bonacina. Towards a unified model of search in theorem proving: subgoal-reduction strategies. *Journal of Symbolic Computation*, 39(2):209–255, 2005.
16. Maria Paola Bonacina. On theorem proving for program checking – Historical perspective and recent developments. In Maribel Fernàndez, editor, *Proceedings of the 12th International Symposium on Principles and Practice of Declarative Programming (PPDP)*, pages 1–11. ACM Press, 2010.

17. Maria Paola Bonacina, Christopher A. Lynch, and Leonardo de Moura. On deciding satisfiability by DPLL($\Gamma + \mathscr{T}$) and unsound theorem proving. In Renate Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 35–50. Springer, 2009.
18. 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.
19. Maria Paola Bonacina and David A. Plaisted. Constraint manipulation in SGGS. In Temur Kutsia and Christophe Ringeissen, editors, *Proceedings of the 28th Workshop on Unification (UNIF)*, Technical Reports of the Research Institute for Symbolic Computation, pages 47–54. Johannes Kepler Universität, July 2014. Available at `http://vsl2014.at/meetings/UNIF-index.html`.
20. Maria Paola Bonacina and David A. Plaisted. SGGS theorem proving: an exposition. In Boris Konev, Leonardo De Moura, and Stephan Schulz, editors, *Proceedings of the 4th Workshop on Practical Aspects in Automated Reasoning (PAAR)*, EasyChair Proceedings in Computing (EPiC), pages 1–14, July 2014.
21. Maria Paola Bonacina and David A. Plaisted. Semantically guided goal-sensitive reasoning: inference system and completeness. In preparation, 2015.
22. Aaron R. Bradley and Zohar Manna. *The Calculus of Computation - Decision Procedures with Applications to Verification*. Springer, 2007.
23. Ricardo Caferra, Alexander Leitsch, and Nicholas Peltier. *Automated Model Building*. Kluwer Academic Publishers, 2004.
24. Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.
25. Heng Chu and David A. Plaisted. Model finding in semantically guided instance-based theorem proving. *Fundamenta Informaticae*, 21(3):221–235, 1994.
26. Heng Chu and David A. Plaisted. CLINS-S: a semantically guided first-order theorem prover. *Journal of Automated Reasoning*, 18(2), 1997.
27. Scott Cotton. Natural domain SMT: A preliminary assessment. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Proceedings of the 8th International Conference on Formal Modeling of Timed Systems (FORMATS)*, volume 6246 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2010.
28. Martin Davis. *The Universal Computer. The Road from Leibniz to Turing*. Mathematics/Logic/Computing Series. CRC Press, Taylor and Francis Group, 2012. Turing Centenary Edition.
29. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.
30. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
31. Leonardo de Moura and Nikolaj Bjørner. Deciding effectively propositional logic using DPLL and substitution sets. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proceedings of the 4th International Conference on Automated Reasoning (IJCAR)*, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 410–425. Springer, 2008.
32. Leonardo de Moura and Nikolaj Bjørner. Engineering DPLL(T) + saturation. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proceedings of the 4th International Conference on Automated Reasoning (IJCAR)*, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 475–490. Springer, 2008.
33. Leonardo de Moura and Nikolaj Bjørner. Bugs, moles and skeletons: Symbolic reasoning for software development. In Jürgen Giesl and Reiner Hähnle, editors, *Proceedings of the 5th International Conference on Automated Reasoning (IJCAR)*, volume 6173 of *Lecture Notes in Artificial Intelligence*, pages 400–411. Springer, 2010.
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 the 14th International Conference on Verification with Model Checking and Abstract Interpretation (VMCAI)*, volume 7737 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2013.
36. David L. Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *Journal of the ACM*, 52(3):365–473, 2005.
37. Niklas Eén and Niklas Sörensson. An extensible SAT solver [extended version 1.2]. In Enrico Giunchiglia and Armando Tacchella, editors, *Proceedings of the 6th International Conference on*

Theory and Applications of Satisfiability Testing (SAT), volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.
38. Moshe Emmer, Zurab Khasidashvili, Konstantin Korovin, Christoph Sticksel, and Andrei Voronkov. EPR-based bounded model checking at word level. In Bernhard Gramlich, Dale Miller, and Ulrike Sattler, editors, *Proceedings of the 6th International Conference on Automated Reasoning (IJCAR)*, volume 7364 of *Lecture Notes in Artificial Intelligence*, pages 210–224. Springer, 2012.
39. Christian Fermüller, Alexander Leitsch, Ulrich Hustadt, and Tanel Tammet. Resolution decision procedures. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 25, pages 1793–1849. Elsevier, 2001.
40. Arnaud Fietzke. *Labelled superposition*. PhD thesis, Max Planck Institut für Informatik, Saabrücken, October 2013.
41. Arnaud Fietzke and Christoph Weidenbach. Labelled splitting. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proceedings of the 4th International Conference on Automated Reasoning (IJCAR)*, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 459–474. Springer, 2008.
42. Harald Ganzinger and Konstantin Korovin. New directions in instantiation-based theorem proving. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS)*, pages 55–64. IEEE Computer Society Press, 2003.
43. Harald Ganzinger and Konstantin Korovin. Theory instantiation. In Miki Hermann and Andrei Voronkov, editors, *Proceedings of the 13th Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 497–511. Springer, 2006.
44. Leopold Haller, Alberto Griggio, Martin Brain, and Daniel Kroening. Deciding floating-point logic with systematic abstraction. In Gianpiero Cabodi and Satnam Singh, editors, *Proceedings of the 12th Conference on Formal Methods for Computer-Aided Design (FMCAD)*. ACM and IEEE, 2012.
45. Krystof Hoder and Andrei Voronkov. The 481 ways to split a clause and deal with propositional variables. In Maria Paola Bonacina, editor, *Proceedings of the 24th International Conference on Automated Deduction (CADE)*, volume 7898 of *Lecture Notes in Artificial Intelligence*, pages 450–464. Springer, 2013.
46. 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.
47. Swen Jacobs and Uwe Waldmann. Comparing instance generation methods for automated reasoning. *Journal of Automated Reasoning*, 38:57–78, 2007.
48. Himanshu Jain. *Verification using satisfiability checking, predicate abstraction and Craig interpolation*. PhD thesis, School of Computer Science, Carnegie Mellon University, 2008.
49. João P. Marques-Silva and Karem A. Sakallah. GRASP: A new search algorithm for satisfiability. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 220–227, 1997.
50. 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 the 13th Conference on Formal Methods in Computer Aided Design (FMCAD)*. FMCAD Inc., 2013.
51. Dejan Jovanović and Leonardo de Moura. Cutting to the chase: Solving linear integer arithmetic. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Proceedings of the 23rd International Conference on Automated Deduction (CADE)*, volume 6803 of *Lecture Notes in Artificial Intelligence*, pages 338–353. Springer, 2011.
52. Dejan Jovanović and Leonardo de Moura. Solving non-linear arithmetic. In Bernhard Gramlich, Dale Miller, and Ulrike Sattler, editors, *Proceedings of the 6th International Conference on Automated Reasoning (IJCAR)*, volume 7364 of *Lecture Notes in Artificial Intelligence*, pages 339–354. Springer, 2012.
53. Konstantin Korovin. An invitation to instantiation-based reasoning: from theory to practice. In Renate Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 163–166. Springer, 2009.
54. Konstantin Korovin. Inst-Gen: a modular approach to instantiation-based automated reasoning. In Andrei Voronkov and Christoph Weidenbach, editors, *Programming Logics: Essays in Memory of Harald Ganzinger*, volume 7797 of *Lecture Notes in Artificial Intelligence*, pages 239–270. Springer, 2013.
55. Konstantin Korovin and Christoph Sticksel. iProver-Eq: An instantiation-based theorem prover with equality. In Jürgen Giesl and Reiner Hähnle, editors, *Proceedings of the 5th International Conference*

on Automated Reasoning (IJCAR), volume 6173 of *Lecture Notes in Artificial Intelligence*, pages 196–202. Springer, 2010.
56. Konstantin Korovin, Nestan Tsiskaridze, and Andrei Voronkov. Conflict resolution. In Ian P. Gent, editor, *Proceedings of the 15th International Conference on Constraint Programming (CP)*, volume 5732 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 2009.
57. Laura Kovàcs and Andrei Voronkov. First order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Proceedings of the 25th International Conference on Computer-Aided Verification (CAV)*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013.
58. Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View*. Springer, 2008.
59. Shie-Jue Lee and David A. Plaisted. Theorem proving using hyper-matching strategy. In Z. Ras, editor, *Methodologies for Intelligent Systems*, pages 467–476. North-Holland, 1989. Presented at the International Symposium on Methodologies for Intelligent Systems, October 1989.
60. Shie-Jue Lee and David A. Plaisted. Inference by clause matching. In Z. Ras and M. Zemankova, editors, *Intelligent Systems*, pages 200–235. Ellis Horwood, 1990.
61. Shie-Jue Lee and David A. Plaisted. Eliminating duplication with the hyperlinking strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.
62. K. Rustan M. Leino and Aleksandar Milicevic. Program extrapolation with Jennisys. In *Proceedings of the 27th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 411–430. ACM, 2012.
63. Reinhold Letz. Clausal tableaux. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction - A Basis for Applications*, volume I: Foundations - Calculi and Methods, chapter 2, pages 43–72. Kluwer Academic Publishers, 1998.
64. Reinhold Letz and Gernot Stenz. DCTP - a disconnection calculus theorem prover. In Rajeev P. Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the 1st International Conference on Automated Reasoning (IJCAR)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 381–385. Springer, 2001.
65. Reinhold Letz and Gernot Stenz. Model elimination and connection tableau procedures. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, chapter 28, pages 2015–2114. Elsevier, 2001.
66. Reinhold Letz and Gernot Stenz. Proof and model generation with disconnection tableaux. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Proceedings of the 8th International Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 142–156. Springer, 2001.
67. Reinhold Letz and Gernot Stenz. Integration of equality reasoning into the disconnection calculus. In Uwe Egly and Christian G. Fermüller, editors, *Proceedings of the 15th International Conference on Analytic Tableaux and Related Methods (TABLEAUX)*, volume 2381 of *Lecture Notes in Artificial Intelligence*, pages 176–190. Springer, 2002.
68. D. W. Loveland. A simplified format for the model elimination procedure. *Journal of the ACM*, 16(3):349–363, 1969.
69. Sharad Malik and Lintao Zhang. Boolean satisfiability: from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76–82, 2009.
70. William W. McCune. Semantic guidance with carefully chosen interpretations. In Myla Archer, Thierry Boy de la Tour, and Cesar Muñoz, editors, *Pre-Proceedings of the 6th Workshop on Strategies in Automated Deduction (STRATEGIES)*, pages 76–83, 2006.
71. Kenneth L. McMillan, A. Kuehlmann, and Mooly Sagiv. Generalizing DPLL to richer logics. In Ahmed Bouajjani and Oded Maler, editors, *Proceedings of the 21st Conference on Computer Aided Verification (CAV)*, volume 5643 of *Lecture Notes in Computer Science*, pages 462–476. Springer, 2009.
72. 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 the 39th Design Automation Conference (DAC)*, pages 530–535, 2001.
73. Juan Antonio Navarro and Andrei Voronkov. Proof systems for effectively propositional logic. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proceedings of the 4th International Conference on Automated Reasoning (IJCAR)*, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 426–440. Springer, 2008.
74. 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.

75. Gerald E. Peterson and Mark E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981.
76. David A. Plaisted. Mechanical theorem proving. In Ranan B. Banerji, editor, *Formal Techniques in Artificial Intelligence*, pages 269–320. Elsevier, 1990.
77. David A. Plaisted and Swaha Miller. The relative power of semantics and unification. In Andrei Voronkov and Christoph Weidenbach, editors, *Programming Logics: Essays in Memory of Harald Ganzinger*, volume 7797 of *Lecture Notes in Artificial Intelligence*, pages 317–344. Springer, 2013.
78. David A. Plaisted and Yunshan Zhu. *The Efficiency of Theorem Proving Strategies*. Friedr. Vieweg & Sohns, 1997.
79. David A. Plaisted and Yunshan Zhu. Ordered semantic hyper linking. *Journal of Automated Reasoning*, 25:167–217, 2000.
80. Alexandre Riazanov. *Implementing an efficient theorem prover*. PhD thesis, Department of Computer Science, The University of Manchester, July 2003.
81. J. Alan Robinson. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1:227–234, 1965.
82. Stephan Schulz. System description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proceedings of the 19th International Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 8312 of *Lecture Notes in Artificial Intelligence*, pages 735–743. Springer, 2013.
83. Natarajan Shankar. Little engines of proof, 2002. Invited talk, Federated Logic Conference, Copenhagen, Denmark; and course notes of Fall 2003, `http://www.csl.sri.com/users/shankar/LEP.html`.
84. James R. Slagle. Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 14(4):687–697, 1967.
85. Raymond M. Smullyan. *First-Order Logic*. Dover Publications, New York, 1995. First published by Springer in 1968.
86. Mark E. Stickel. A unification algorithm for associative commutative functions. *Journal of the ACM*, 28:423–434, 1981.
87. Mark E. Stickel. A Prolog technology theorem prover. *New Generation Computing*, 2(4):371–383, 1984.
88. Mark E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.
89. Mark E. Stickel. A Prolog technology theorem prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.
90. Mark E. Stickel. A Prolog technology theorem prover: new exposition and implementation in Prolog. *Theoretical Computer Science*, 104:109–128, 1992.
91. Allen Van Gelder and Yumi K. Tsuji. Satisfiability testing with more reasoning and less guessing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:559–586, 1996.
92. Christoph Weidenbach. Combining superposition, sorts and splitting. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, pages 1965–2012. Elsevier, 2001.
93. Christoph Weidenbach, Dylana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In Renate Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 140–145. Springer, 2009.
94. 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.
95. Hantao Zhang and Mark E. Stickel. Implementing the Davis-Putnam method. *Journal of Automated Reasoning*, 24(1/2):277–296, 2000.
96. Lintao Zhang and Sharad Malik. The quest for efficient boolean satisfiability solvers. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE)*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 295–313. Springer, 2002.