SGGS Theorem Proving: an Exposition

Maria Paola Bonacina[1] and David A. Plaisted[2]

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

[2] Department of Computer Science, UNC at Chapel Hill, USA
plaisted@cs.unc.edu

Abstract

We present in expository style the main ideas in SGGS, which stands for Semantically-Guided Goal-Sensitive theorem proving. SGGS uses sequences of constrained clauses to represent models, instance generation to go from a candidate model to the next, and resolution as well as other inferences to repair the model. SGGS is *refutationally complete* for first-order logic, *DPLL-style model based*, *semantically guided*, *goal sensitive*, and *proof confluent*, which appears to be a rare combination of features. In this paper we describe the core of SGGS in a narrative style, emphasizing ideas and trying to keep technicalities to a minimum, in order to advertise it to builders and users of theorem provers.

Introduction

The problem of first-order refutational clausal theorem proving could be considered solved, because many refutationally complete methods are known. Motivations for seeking new methods come from the practice. For the work presented in this paper, the motivation resides in recognizing the importance of having a first-order method that is *simultaneously* DPLL-style model-based, semantically guided, goal sensitive, and proof confluent. We describe a method, called SGGS, which has all these properties. The presentation aims at being simple and informal, leaving to other papers the technical details.

The recognition of the importance of each of these characteristics is of course not new. *Semantic guidance* by a given interpretation is desirable, because the search space of a first-order problem is infinite. *Goal sensitivity*, which means generating clauses connected, in some sense, with the goal to be refuted, is similarly desirable, especially if there are many axioms or a large knowledge base. *Proof confluence*, which means that no step prevents the system from finding a proof, so that backtracking is not needed, is desirable, because first-order problems may cause a lot of backtracking. A method is *DPLL-style model based*, if the state of the derivation includes the description of a candidate (partial) model, inference and search for a model help each other, and inference is seen (also) as *model transformation*. In SGGS, semantic guidance and model-based style are connected, because the given interpretation is also the initial candidate model. The DPLL renaissance, *conflict-driven clause learning* (CDCL), understood as inference and search guiding each other, the practical success of model-based SAT and SMT solvers, and the observation that models are relevant to applications and intuitive for users, have all contributed to make such a model-based style appealing (e.g., [32, 18] for introductory articles). However, model-based first-order reasoning is challenging. One way to go about it is to generalize features such as CDCL from propositional reasoning to reasoning in first-order theories (e.g., [19, 24]). Another way is to approach the problem in generic first-order theorem proving.

Among the paradigms for first-order refutational clausal theorem proving, resolution is natively proof confluent. The idea of semantic guidance produces, for instance, *hyperresolution* and *semantic resolution*; and that of goal sensitivity yields, for example, *resolution with set of* *support*; *linear resolution* is even more goal sensitive, but at the expense of proof confluence. However, resolution is not model-based: the models remain implicit, and play a rôle only in the proof of refutational completeness, where *semantic trees* are used to survey models and show that the inference system excludes them all. These remarks apply also to the methods based on resolution and paramodulation/superposition for first-order logic with equality, using *transfinite semantic trees* [22], or the *rewrite model* [1]. The extension of SGGS to first-order logic with equality is future work.

Model elimination and tableaux-based methods can be considered model based, because they represent models by chains of literals or branches in tableaux, and weakly goal-sensitive, because one can start the tableaux with a clause coming from the negation of the conjecture, as in linear resolution. However, they are not proof confluent. This dichotomy between resolution (proof confluent but not model based) and model elimination tableaux (model based but not proof confluent) led to investigate *hybrid strategies* that combine *instance generation*, as suggested by the Herbrand theorem, with tableaux, as in the *disconnection calculus* [10, 29, 30, 31], or *hypertableaux* [2, 9] (cf. Section 7.3 of [11] for a discussion of these strategies). These methods are model based in the way tableaux-based methods are, and proof confluent, because they generate instances rather than instantiate the variables in tableaux. In this way they avoid backtracking, which is needed in tableaux to undo the instantiation of rigid variables. However, these methods are not semantically guided, and not goal sensitive, since they link clauses to the tableau irrespective of the goal.

The quest continued with methods that combine instance generation with a DPLL-style model-based approach: the procedure maintains a candidate model, generates ground instances false in the candidate model, updates it to satisfy them, and continues until either it finds a model or proves unsatisfiability. The aim is to let model search guide instance generation. While the origins of this perspective may be traced back to *hyperlinking-based methods* [16, 17, 34], and the already mentioned disconnection calculus and hypertableaux, the notion of generalizing features of DPLL to instance-based methods gave rise to FDPLL [3], for *First-order DPLL*, its successor the *model-evolution calculus* [4, 5, 7, 8, 6], and the *Inst-Gen method* [20, 21, 27, 26].

The problem is how to realize this DPLL-style model-based scheme in a way that is first order, semantically guided, goal sensitive, and proof confluent. For instance, *ordered semantic hyperlinking* [34] is semantically guided and model based, but not first order; the model-evolution calculus is first order and model based, but not goal sensitive and not proof confluent. In this paper we describe the core of SGGS in a narrative style, emphasizing ideas and trying to keep technicalities to a minimum, in order to advertise it to builders and users of automated reasoners. The representation of models by SGGS clause sequences is studied in [14]. The constraint solving part is covered in [13]. A manuscript including all aspects of the method, with the technical details, the proofs of refutational completeness and goal-sensitivity, and more references and comparison with related work is available as [15].

Clause Sequences, Models, and Derivations

While in propositional logic a (partial) model may be represented by a sequence of literals, for first-order logic SGGS uses a sequences of *constrained clauses* $A \rhd C$, where A is a constraint and C is a clause. A *constraint* is either an atomic constraint, or the negation, conjunction, or disjunction of constraints. An *atomic constraint* may be empty, denoted by *true*, so that plain clauses are included as a special case, or an expression of the form $x \equiv y$ or $top(t) = f$, where x and y are variables, $\equiv$ is syntactic identity, f is a function symbol, t is a term, and $top(t)$ is the top symbol of term t. A variable that appears in A but not in C is implicitly existentially quantified. A constraint is in *standard form*, if it is 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 paper all constraints are in standard form: standardization is covered in [13, 15].

A key feature of both model representation and inference in SGGS is *literal selection*: 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 selected in C, and C' is an instance of C, the literal of C' that is instance of L is selected in C'. Just like a clause represents its ground instances, a constrained clause $A \rhd C$ represents its *constrained ground instances* (cgi's), that are the ground instances of C that satisfy the constraint A. Thus, the set of cgi's of $A \rhd C$ is $Gr(A \rhd C) = \{C\vartheta : \ \models A\vartheta,\ C\vartheta \ ground\}$, and $Gr(A \rhd L)$ is defined in the same way. For example, $P(a, b) \in Gr(x \not\equiv y \rhd P(x, y))$, but $P(b, b) \notin Gr(x \not\equiv y \rhd P(x, y))$.

SGGS Clause Sequences

SGGS is semantically guided by a fixed interpretation I. A constrained literal $A \rhd L$ is *I-true* if all its cgi's are true in I, that is, $I \models Gr(A \rhd L)$. Since variables are implicitly universally quantified, in order to falsify a constrained literal, it is sufficient to falsify one of its cgi's. SGGS uses a stronger notion of falsification: a constrained literal $A \rhd L$ is *I-false*, if all its cgi's are false in I, that is, the *complementary literals* of all its cgi's are true in I. A constrained literal may be neither I-true nor I-false, because it can be that some of its cgi's are true in I and some are false. Note that for ground literals, to be I-true and true in I is the same thing, and to be I-false and false in I is the same thing, because a ground literal has only one cgi, which is the literal itself.

For clauses, we say that a constrained clause 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: a clause is true in I if all its ground instances are; that is, for all ground instances, there is at least one literal which is true in I. On the other hand, a clause is I-all-true if all ground instances of all its literals are true in I. Similarly, being I-all-false is stronger than being false in I: a clause is false in I if it has a ground instance that is false in I; that is, all literals in this ground instance are false in I. On the other hand, a clause is I-all-false if all ground instances of all its literals are false in I.

The basic structure that SGGS works with is the *clause sequence*: it is either empty, denoted by ε, or a finite sequence of constrained clauses $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$ such that for all i, $1 \leq i \leq n$:

1. Every literal in C_i is either I-true or I-false;
2. A literal L_i in C_i is selected; and
3. If a clause has I-false literals, then one is selected.

Models

How does a clause sequence represent a partial interpretation? The *partial interpretation* $I^p(\Gamma)$ induced by a clause sequence Γ is defined inductively over the length of the clause sequence, in such a way that each constrained clause in the sequence may contribute. Thus, we need the notion of prefix: given Γ as above, its *prefix* of length j, denoted $\Gamma|_j$, for $1 \leq j \leq n$, is $A_1 \rhd C_1[L_1], \ldots, A_j \rhd C_j[L_j]$. Note that $\Gamma|_n$ is Γ itself.

If Γ is empty, its induced partial interpretation is also empty: $I^p(\varepsilon) = \emptyset$. Otherwise, we define $I^p(\Gamma|_i)$, for all i, $0 < i \leq n$, and $I^p(\Gamma)$ is $I^p(\Gamma|_n)$. $I^p(\Gamma|_i)$ is $I^p(\Gamma|_{i-1})$ plus the *proper constrained ground instances* (pcgi's) of the selected literal L_i in clause $A_i \rhd C_i[L_i]$. The pcgi's of L_i are simply its instances in the pcgi's of $A_i \rhd C_i[L_i]$. The pcgi's of $A_i \rhd C_i[L_i]$ are those cgi's that have no intersection with $I^p(\Gamma|_{i-1})$, and whose selected literal does not appear negated in $I^p(\Gamma|_{i-1})$. The first requirement means that these cgi's are not satisfied by $I^p(\Gamma|_{i-1})$. The second requirement means that the selected literals of these cgi's can be safely added to $I^p(\Gamma|_{i-1})$ to form $I^p(\Gamma|_i)$, because their complements do not appear in $I^p(\Gamma|_{i-1})$.

$I^p(\Gamma)$ can be completed in an interpretation, denoted $I[\Gamma]$, by consulting I, whenever $I^p(\Gamma)$ does not determine the truth of a ground literal: the *interpretation* $I[\Gamma]$ *induced* by Γ is I modified to satisfy the proper constrained ground instances of the selected literals in Γ.

For example, let I be the interpretation that makes all negative literals true. If Γ is the sequence of unit clauses $P(a, x), P(b, y), \neg P(z, z), P(u, v)$, 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. Note how $I[\Gamma]$ is built by consulting Γ left to right, so that the order of clauses in a sequence is meaningful. This is a consequence of having defined $I^p(\Gamma)$ by induction on the length of the sequence by using prefixes.

For another example, again with I all negative, let $\Gamma = C_1, C_2, C_3$ be $[P(x)]$, $\neg P(f(y)) \vee [Q(y)]$, $\neg P(f(z)) \vee \neg Q(g(z)) \vee [R(f(z), g(z))]$, where selected literals are in brackets. $I[\Gamma|_1]$ 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 $R(f(z), g(z))$.

SGGS Derivations

Although a clause sequence is the main component of the *state* of a derivation in SGGS, there is also some other information that SGGS needs to maintain, and that is an *assignment* of I-true literals to I-false literals. Intuitively, since we are looking for a refutation, I-true literals need to be neutralized, by showing that they are covered, in some sense, by I-false literals. However, this needs to apply to I-true literals that are *not selected*: if $A_i \rhd C_i[L_i]$ has an I-true literal L other than L_i, the focus on L_i is spoiled, in a sense, because the clause is satisfied already by I with another literal, namely L. Thus, such an I-true literal L must be assigned to a preceding clause, that is an $A_j \rhd C_j[L_j]$ with $j < i$, such that L_j is I-false, and all cgi's of $A_i \rhd L$ appear with opposite sign among the pcgi's of $A_j \rhd L_j$. It follows that L is false in $I^p(\Gamma|_j)$, $I[\Gamma|_j]$ and $I[\Gamma]$, and false because all its cgi's are, and not just one. We say that $A_i \rhd C_i[L_i]$ *depends* on $A_j \rhd C_j[L_j]$. It is plausible that an assignment thus defined is a feature of SGGS geared towards refutation, and as such, it may be one of the feature to be re-assessed if one wanted to orient the method towards model finding. From now on, an *SGGS clause sequence* is a clause sequence equipped with such an assignment for every one of its clauses.

Given a set S of clauses and an 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 Γ_0 is empty, and Γ_j with $j > 0$ is generated from Γ_{j-1}, S, and I, by an SGGS inference rule. A derivation is a *refutation* if there is a $k > 0$ such that Γ_k contains the empty clause, written $\perp$.

A propositional example

We consider an example to get an intuition of how SGGS behaves at the propositional level. We compare SGGS with a version of DPLL applied to $S = \{P \vee Q, P \vee \neg Q, \neg P \vee Q, \neg P \vee \neg Q\}$

with $P < Q$ in an ordering on atoms and initial interpretation $I = \{\neg P, \neg Q\}$. Clause $P \vee Q$ contradicts I. Then I is modified to $I_1 = \{\neg P, Q\}$ to satisfy $P \vee Q$. Clause $P \vee \neg Q$ contradicts I_1. Now these two clauses are resolved to produce P. Next I_1 is modified to $I_2 = \{P, Q\}$ to satisfy P. Clause $\neg P \vee \neg Q$ contradicts I_2. Thus I_2 is modified to $I_3 = \{P, \neg Q\}$ to satisfy P and $\neg P \vee \neg Q$. Clause $\neg P \vee Q$ contradicts I_3. This clause is resolved with $\neg P \vee \neg Q$ to yield $\neg P$. Finally, $\neg P$ resolves with P to give $\perp$.

The behavior of SGGS is both similar and different. Also in SGGS the search is guided by the semantics, and unsatisfiability is shown by finding clauses that contradict each interpretation. In the above DPLL example, resolution is applied to already chosen clauses that contradict two interpretations differing in maximal literals, which are resolved upon; and I is modified each time to satisfy a maximal literal of a clause. In SGGS the rôle of maximal literals is played by selected literals: at each stage $I[\Gamma]$ is I modified to satisfy selected literals; only selected literals are resolved upon; since every literal in an SGGS sequence is either I-true or I-false, resolution resolves an I-true with an I-false literal; one of the parents must be I-all-true, and precede the other in the sequence. Thus, clauses may be reordered prior to resolution. After a resolution step, the parent that is not I-all-true is removed, together with all the clauses depending on it. Figure 1 shows an SGGS refutation of S with I as in the DPLL example, labelling $C_1, \ldots, C_i, \ldots$ the clauses in the current sequence.

Γ_0.	ε	$I[\Gamma_0] = I = \{\neg P, \neg Q\}$
Γ_1.	$Q \vee [P]$	(SGGS-extension) $I[\Gamma_1] = \{P, \neg Q\}$
		P and Q are both I-false;
		C_1 contradicts $I[\Gamma_0]$ but $I[\Gamma_1]$ satisfies it.
Γ_2.	$Q \vee [P],\ \neg P \vee [Q]$	(SGGS-extension) $I[\Gamma_2] = \{P, Q\}$
		In C_2, $\neg P$ is I-true, assigned to C_1; Q is I-false;
		C_2 contradicts $I[\Gamma_1]$; $I[\Gamma_2]$ satisfies C_1 and C_2.
Γ_3.	$Q \vee [P],\ \neg P \vee [Q],\ \neg P \vee [\neg Q]$	(SGGS-extension) $I[\Gamma_3] = \{P, Q\}$
		In C_3, $\neg P$ and $\neg Q$ are I-true and assigned to C_1 and C_2, respectively.
Γ_4.	$Q \vee [P],\ \neg P \vee [\neg Q],\ \neg P \vee [Q]$	(Reordering before resolution) $I[\Gamma_4] = \{P, \neg Q\}$
		In C_2, $\neg Q$ is not assigned.
Γ_5.	$Q \vee [P],\ \neg P \vee [\neg Q],\ [\neg P]$	(Resolving C_2 and C_3 in Γ_4) $I[\Gamma_5] = \{P, \neg Q\}$
Γ_6.	$[\neg P],\ Q \vee [P],\ \neg P \vee [\neg Q]$	(Reordering before resolution) $I[\Gamma_6] = \{\neg P, \neg Q\}$
		In C_1, $\neg P$ is not assigned.
Γ_7.	$[\neg P],\ [Q],\ \neg P \vee [\neg Q]$	(Resolving C_1 and C_2 in Γ_6) $I[\Gamma_7] = \{\neg P, Q\}$
		C_3 has nowhere to assign $\neg P$.
Γ_8.	$[\neg P],\ [Q]$	(Deletion of $\neg P \vee \neg Q$ that depended on $Q \vee P$)
		$I[\Gamma_8] = \{\neg P, Q\}$
Γ_9.	$[\neg P],\ [Q],\ \neg Q \vee [P]$	(SGGS-extension) $I[\Gamma_9] = \{\neg P, Q\}$
		In C_3, $\neg Q$ is I-true and assigned to C_2.
Γ_{10}.	$[\neg P],\ [Q],\ [\neg Q]$	(Resolving C_1 and C_3 in Γ_9) $I[\Gamma_{10}] = \{\neg P, Q\}$
Γ_{11}.	$[\neg P],\ [\neg Q],\ [Q]$	(Reordering before resolution) $I[\Gamma_{11}] = \{\neg P, \neg Q\}$
Γ_{12}.	$[\neg P],\ [\neg Q],\ \perp$	(Resolving C_2 and C_3 in Γ_{11})

Figure 1: Example of a refutation by SGGS in propositional logic.

The Main Inference Rules

The three main inference mechanisms in SGGS are *instance generation* by extension, a restricted form of *resolution*, and a few *splitting rules* to *partition* clauses. *SGGS-extension* adds to the current clause sequence an instance of a clause in S: the objective is to find a model of all instances of all clauses in S, and if some are not covered, they must be added. It may happen that selected literals have ground instances in common. If the literals have opposite sign, this would make the model *inconsistent*: SGGS features a restricted form of resolution, called *SGGS-resolution*, to remove such contradictions. SGGS-resolution represents an implicit sort of backtracking over the set of possible models of S. The resolvent is a *lemma*, that constrains the model, because the model must satisfy it, and intuitively captures a portion of the search space of models that has been explored. If resolution generates the empty clause, no model can be found. As usual, SGGS-resolution is accompanied by a form of factoring, called *SGGS-factoring*. If selected literals have ground instances in common, and have the same sign, there is *duplication*. SGGS features *splitting rules* that partition a clause with respect to another clause. The clause that gets partitioned, or split, is replaced by other clauses, that have its same set of ground instances, in such a way that the duplicated literals are isolated and can be removed.

SGGS-extension

The idea of SGGS-extension is that if there is a ground instance $D \equiv C\mu$ (as usual, lower case Greek letters denote substitutions) of a clause C in S that is not satisfied by the interpretation $I[\Gamma]$ induced by the current Γ, then Γ should be amended to satisfy D. SGGS works at the first-order level, not at the ground level. Therefore, it captures D by generating a constrained clause E, such that $E \equiv C\vartheta$ is an instance of C, and $D \equiv C\vartheta\tau \equiv E\tau$ is an instance of E. Adding E to Γ modifies $I[\Gamma]$ to satisfy D. The conflict between D, hence E, and $I[\Gamma]$ resembles a conflict in DPLL, and therefore an SGGS-extension step represents a fresh start, with an effect similar to that of backtracking in DPLL.

To understand the mechanics of SGGS-extension we reason as follows. Let $D \equiv C\mu$ be a ground instance of a clause C in S such that $I[\Gamma] \not\models D$. This means that all literals of D are false in $I[\Gamma]$. Assume that they are also all false in I. Then, it is sufficient to extend Γ with an instance $C\beta$ of C, such that D is an instance of $C\beta$, and β is a most general substitution such that all and only the cgi's of $C\beta$ are false in I (which implies $C\beta$ is I-false). We call such a substitution β a *most general I-falsifier*. Note that β may also be empty, and in such a case we simply extend the sequence with C. This typically happens at the beginning of the derivation, when we simply extend Γ with I-false input clauses (e.g., the first two steps in Figure 2).

Assume instead that some literals in D are true in I. Let $L_1, \ldots, L_n$, $n > 0$, be the literals of C such that $L_1\mu, \ldots, L_n\mu$ are the literals of D that are true in I. Thus, these literals $L_1\mu, \ldots, L_n\mu$ are true in I and false in $I[\Gamma]$. Since $I[\Gamma]$ differs from I for the pcgi's of selected literals, this means that there must be clauses $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$ in Γ, such that $M_1, \ldots, M_n$ have pcgi's, $I[\Gamma] \models M_i$, and $L_i\mu$ is an instance of $\neg M_i$, for $i = 1, \ldots, n$ (possibly up to some permutation of the indices). It follows that the L_i's and M_i's unify and have opposite sign: $L_i\alpha \equiv \neg M_i\alpha$, for $i = 1, \ldots, n$, where α is their simultaneous most general unifier (mgu).

When we perform inferences at the first-order level, we do not know the ground instance D of C that is falsified by $I[\Gamma]$. Thus, SGGS-extension looks for a clause C in S and clauses $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$ $(n > 0)$ in Γ, such that the literals $M_1, \ldots, M_n$ are I-false, and simultaneously unifiable with n distinct literals $\{L_1, \ldots, L_n\}$ in C having opposite sign: $L_i\alpha \equiv \neg M_i\alpha$, for $1 \leq i \leq n$. As before, α is the simultaneous mgu of these literals. Note how the fact that $M_1, \ldots, M_n$ are I-false means that their pcgi's are true in $I[\Gamma]$, which means that $L_1\alpha, \ldots, L_n\alpha$ are false in $I[\Gamma]$ and "cover" the $L_1\mu, \ldots, L_n\mu$ false in $I[\Gamma]$ of the above reasoning at the ground level. Thus, the clause E that is an instance of C and such that D is an instance of E, will be an instance of $C\alpha$.

Applying to C the mgu α is not enough, though, in general, because $L_1, \ldots, L_n$ are not the only literals in C (and in its ground instance D falsified by $I[\Gamma]$ that we want to capture). In other words, we are not interested in adding to Γ an instance of C that is true in $I[\Gamma]$ thanks to some literal other than $L_1, \ldots L_n$, because such an instance would not capture the ground instance(s) D false in $I[\Gamma]$. Also, the inference rule needs to cover also the case mentioned at the beginning, where D has no I-true literals. Thus, SGGS-extension applies a *most general I-falsifier* β of all the literals in $C\alpha$ other than $L_1\alpha, \ldots, L_n\alpha$.

Since constraints are inherited, SGGS-extension adds the clause $E \equiv (\bigwedge_{j=1}^{n} B_j\vartheta) \rhd C[L]\vartheta$, where $\vartheta = \alpha\beta\gamma$. We omit for simplicity the technical explanation of the third substitution γ, named *extension substitution*. In practice, it is an mgu of I-false literals in $C\alpha\beta$ with the complements of selected I-true literals in Γ. Its purpose is to make the added instance as precise as possible in capturing ground instances falsified by $I[\Gamma]$. Note that every literal in $C\alpha\beta$ is either I-true or I-false by construction: the $L_i\alpha\beta$'s are I-true (because $L_i\alpha \equiv \neg M_i\alpha$, for $1 \leq i \leq n$, and the M_i's are required to be I-false), and all other literals in $C\alpha\beta$ are I-false (because β is a most general I-falsifier for them). Thus, every literal in E is either I-true or I-false, as required by the definition of clause sequence. This also means that an assignment for clause E simply assigns its I-true literals $L_1\vartheta, \ldots, L_n\vartheta$ to the side premises $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$ of the SGGS-extension step.

Other technical aspects of SGGS-extension include the choice of selected literal in the added clause E, and the placement of E in Γ. For simplicity, we can think of the added clause E as being placed always at the rightmost end of Γ, so that an SGGS-extension step transforms Γ into ΓE. There is an exception to this, which is connected to the proof of refutational completeness. The proof employs a *well-founded total ordering on ground literals*, applied to compare pcgi's of selected literals. For convergence under this ordering, there is a situation where SGGS-extension places E to the left of other clauses with larger selected literals. However, these technicalities are beyond an expository paper, and we refer the interested reader to [15].

In terms of implementation, finding literals that unify and have opposite signs, computing mgu's, and using a well-founded total ordering on ground literals are routine operations for theorem provers. In addition, SGGS-extension requires evaluation with respect to the given interpretation I to find I-false literals, and computation of a most general I-falsifier for a bunch of literals. However, I is fixed; when I is determined by sign as in hyperresolution (e.g., all-positive, or all-negative, as in the manual examples in this paper), the sign of literals determines whether they are I-false or I-true.

SGGS-splitting

Then SGGS features a few inference rules that *split* a clause occurring in the current SGGS clause sequence Γ with respect to another clause in Γ. The point is that the selected literals of clauses in Γ may have cgi's in common, meaning that they have the same atoms. We use $at(L)$ for the atom of literal L, and $at(T) = \{at(L) \,:\, L \in T\}$ for the atoms of a set T of literals. Then we say that constrained literals $A \rhd L$ and $B \rhd M$ *intersect* if $at(Gr(A \rhd L)) \cap at(Gr(B \rhd M)) \neq \emptyset$, and are *disjoint*, otherwise. If $A \rhd L$ and $B \rhd M$ do not share variables, they intersect if and only if $at(L)$ and $at(M)$ unify and the constraint $(A \wedge B)\sigma$ is satisfiable, where σ is the mgu of $at(L)$ and $at(M)$. The intersection is given by $at(Gr(A \rhd L)) \cap at(Gr(B \rhd M)) = at(Gr((A \wedge B)\sigma \rhd M\sigma)) = Gr((A \wedge B)\sigma \rhd at(M)\sigma)$.

Having a notion of intersection, and therefore one of disjointness, we can capture the idea of a partition of a clause: a *partition* of a constrained clause $A \rhd C[L]$ is a set of constrained clauses $\{A_i \rhd C_i[L_i]\}_{i=1}^{n}$ such that:

1. They have the same cgi's (in symbols, $Gr(A \rhd C) = \bigcup_{i=1}^{n}\{Gr(A_i \rhd C_i[L_i])\}$);
2. The $A_i \rhd L_i$'s are pairwise disjoint; and
3. The L_i's are chosen consistently with L.

For example, $\{\mathit{true} \rhd P(f(z), y),\ \mathit{top}(x) \neq f \rhd P(x, y)\}$ is a partition of $\mathit{true} \rhd P(x, y)$ (which can of course be written simply $P(x, y)$).

If clauses $A \rhd C[L]$ and $B \rhd D[M]$ in an SGGS clause sequence have selected literals L and M that intersect, SGGS features inference rules that replace $A \rhd C[L]$ by $\mathit{split}(C, D)$, that is a partition of $C[L]$, where all cgi's of L that are also cgi's of M are isolated in one of the clauses of the partition. To be precise, a *splitting* of $A \rhd C[L]$ by $B \rhd D[M]$, denoted $\mathit{split}(C, D)$, is a partition $\{A_i \rhd C_i\langle L_i\rangle\}_{i=1}^{n}$ of $A \rhd C[L]$ such that:

1. There is a j, $1 \leq j \leq n$, such that $\mathit{at}(Gr(A_j \rhd L_j)) \subseteq \mathit{at}(Gr(B \rhd M))$, and
2. For all i, $1 \leq i \neq j \leq n$, $\mathit{at}(Gr(A_i \rhd L_i))$ and $\mathit{at}(Gr(B \rhd M))$ are disjoint.

Clause C_j is the *representative* of D in $\mathit{split}(C, D)$: $\mathit{at}(Gr(A_j \rhd L_j))$ is the intersection of $A \rhd L$ and $B \rhd M$.

Computing $\mathit{split}(C, D)$ introduces constraints, including non-standard ones, even when C and D have empty constraints to begin with, and this is precisely why SGGS works with constrained clauses. For example, a splitting of $\mathit{true} \rhd P(x, y)$ by $\mathit{true} \rhd P(f(w), g(z))$ is $\{\mathit{true} \rhd P(f(w), g(z)),\ \mathit{top}(x) \neq f \rhd P(x, y),\ \mathit{top}(y) \neq g \rhd P(f(x), y)\}$. Constraint manipulation to compute splittings is covered in [13, 15].

SGGS-resolution

The third main rule in the SGGS inference system is *SGGS-resolution*: resolution in SGGS applies to two clauses in the current clause sequence Γ. Since the sequence represents a model, resolution is conceived as a way to reason *in* the current model, which corroborates the *model-based* character of the method. We already saw several characteristics of SGGS-resolution in the description of the propositional example of Figure 1: it resolves an I-true selected literal in an I-all-true clause with an I-false selected literal of a following clause in the sequence, and the resolvent *replaces* the parent that is not I-all-true. It is an essential character of the resolution principle to resolve two literals that cannot be simultaneously true in any interpretation. Since SGGS is guided by the given interpretation I, and all literals in a clause sequence are either I-true or I-false, SGGS-resolution resolves an I-true and an I-false literal. Since SGGS selects an I-false literal as soon as a clause has any, an I-true literal can be selected only in an I-all-true clause, and therefore one of the two parents of an SGGS-resolution must be I-all-true.

We are left to see the first-order features of SGGS-resolution, and to explain why the I-all-true parent precedes the other parent in the sequence and why the resolvent replaces the other parent. Intuitively, the reason is that the purpose of resolution in SGGS is not to infer a new clause and add it to a set of clauses, as in typical resolution-based methods, but to *amend* the current model, to improve its representation as offered by the clause sequence. Specifically, assume that $B \rhd D[M]$ precedes $A \rhd C[L]$ in the SGGS clause sequence, $B \rhd D[M]$ is I-all-true, L is I-false, $L \equiv \neg M\vartheta$ for some substitution ϑ, and A logically implies $B\vartheta$. Then the *constrained resolvent* $A \rhd R$, where R is $(C \setminus \{L\}) \cup (D \setminus \{M\})\vartheta$ replaces $A \rhd C[L]$ in the sequence. Note that SGGS-resolution uses *matching*, not unification: the I-false literal L is an instance of the complement of the I-true literal M. This condition ensures (cf. Lemma 5.1 in [15]) that $A \rhd L$

has no pcgi's, because the atoms of the cgi's of $A \rhd L$ that $A \rhd C[L]$ would capture in its position in the sequence are already covered by $B \rhd D[M]$. Therefore, $A \rhd C[L]$ is replaced by $A \rhd R$, which captures the cgi's of $C \setminus \{L\}$. Since the pcgi's of selected literals are what make $I[\Gamma]$, on top of the initial interpretation I, this shows in which sense SGGS-resolution amends the representation of the current model.

Deletion of Disposable Clauses

SGGS-extension adds one clause, splitting rules replace a clause by several, and SGGS-resolution replaces a clause by another one. Does the method ever remove clauses from the sequence? Yes, all constrained clauses that are *disposable* can be removed. This is the task of an inference rule called *deletion*. A non-empty clause is disposable if it has neither proper constrained ground instances nor *complementary constrained ground instances* (ccgi's).

Complementary constrained ground instances are the dual of proper constrained ground instances: the ccgi's of $A_i \rhd C_i[L_i]$ are those cgi's that have no intersection with $I^p(\Gamma|_{i-1})$, and whose selected literal *appears* negated in $I^p(\Gamma|_{i-1})$. Similar to pcgi's, by the first requirement, ccgi's are not satisfied by $I^p(\Gamma|_{i-1})$. The second requirement is the negation of that for pcgi's: the selected literal of a ccgi is the complement of a literal in $I^p(\Gamma|_{i-1})$. Then, the ccgi's of L_i are simply its instances in the ccgi's of $A_i \rhd C_i[L_i]$. In essence, the pcgi's of L_i are those that can be added to the current model, whereas the ccgi's are those that contradict the current model. For lack of space we cannot discuss the rôle of ccgi's further; however, a constrained clause that has neither pcgi's nor ccgi's is useless for the search of a model, and therefore such a clause can be discarded.

When a clause is removed, because it is disposable, or because it is replaced by its resolvent in SGGS-resolution, all the clauses that depend on it according to the assignment are also deleted. Furthermore, when a splitting rule replaces a clause $A \rhd C[L]$ by its splitting $split(C, D)$ with respect to another clause $B \rhd D[M]$ in the sequence, it is possible to delete some of the clauses in $split(C, D)$, provided the representative of D is kept. While the deletion of disposable clauses is part of the inference system, these additional deletions are heuristic at this stage.

We conclude this description of the inference system with an example at the first-order level: Figure 2 shows a refutation for $S = \{\neg P(f(x)) \vee \neg Q(g(x)) \vee R(x),\ P(x),\ Q(y),\ \neg R(c)\}$ with I all-negative.

Refutational Completeness and Goal Sensitivity

An SGGS derivation terminates either with a sequence including the empty clause, or with a sequence which no inference rule can be applied to. We proved that SGGS is *refutationally complete*: if S is unsatisfiable, any *fair* SGGS derivation generates a clause sequence containing the empty clause; if S is satisfiable, the derivation may be infinite, and its limit represents a model of S. The notion of fairness defined for SGGS derivations basically states that any inference, that is infinitely often possible, is eventually done. The argument of the proof of refutational completeness rests on a *well-founded convergence ordering* on clauses sequences, which in turn uses the already mentioned well-founded total ordering on ground literals. By well-foundedness, a derivation that is a non-ascending chain, admits limit, in the form of a *limiting sequence*. We prove that a *bundled* derivation, which satisfies a few simple restrictions, forms a non-ascending chain, and therefore has a limiting sequence. Then we show that if S is unsatisfiable, a fair bundled derivation terminates with a contradiction, because if it does not, there is always an inference that modifies the limit, contradicting well-foundedness.

$\Gamma_0.$	ε	
$\Gamma_1.$	$[P(x)]$	(SGGS-extension)
$\Gamma_2.$	$[P(x)],\ [Q(y)]$	(SGGS-extension)
$\Gamma_3.$	$[P(x)],\ [Q(y)],\ \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)]$	(SGGS-extension)
$\Gamma_4.$	$[P(x)],\ [Q(y)],\ \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$	(SGGS-extension)
$\Gamma_5.$	$[P(x)],\ [Q(y)],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],$ $\neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)],\ [\neg R(c)]$	(split third clause)
$\Gamma_6.$	$[P(x)],\ [Q(y)],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],$ $[\neg R(c)],\ \neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)]$	(move last clause left)
$\Gamma_7.$	$[P(x)],\ [Q(y)],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],$ $[\neg R(c)],\ \neg P(f(c)) \vee [\neg Q(g(c))]$	(resolve last two clauses)
$\Gamma_8.$	$[P(x)],\ \mathit{top}(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],$ $[Q(g(c))],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],$ $[\neg R(c)],\ \neg P(f(c)) \vee [\neg Q(g(c))]$	(split second clause)
$\Gamma_9.$	$[P(x)],\ \mathit{top}(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],$ $\neg P(f(c)) \vee [\neg Q(g(c))],\ [Q(g(c))],$ $x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$	(move last clause left)
$\Gamma_{10}.$	$[P(x)],\ \mathit{top}(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],$ $\neg P(f(c)) \vee [\neg Q(g(c))],\ [\neg P(f(c))],$ $x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$	(resolve clauses 4 and 5)
$\Gamma_{11}.$	$\mathit{top}(x) \neq f \rhd [P(x)],\ y \not\equiv c \rhd [P(f(y))],\ [P(f(c))],$ $\mathit{top}(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],$ $\neg P(f(c)) \vee [\neg Q(g(c))],\ [\neg P(f(c))],$ $x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$	(split first clause)
$\Gamma_{12}.$	$\mathit{top}(x) \neq f \rhd [P(x)],\ y \not\equiv c \rhd [P(f(y))],\ [\neg P(f(c))],$ $[P(f(c))],\ \mathit{top}(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],$ $\neg P(f(c)) \vee [\neg Q(g(c))],$ $x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$	(move $\neg P(f(c))$ left)
$\Gamma_{13}.$	$\mathit{top}(x) \neq f \rhd [P(x)],\ y \not\equiv c \rhd [P(f(y))],\ [\neg P(f(c))],$ $\bot,\ \mathit{top}(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],$ $\neg P(f(c)) \vee [\neg Q(g(c))],$ $x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$	(resolve clauses 3 and 4)

Figure 2: Example of a refutation by SGGS in first-order logic.

Another main result about SGGS is that it is *goal sensitive*, provided the initial interpretation I satisfies the input clauses issued from the assumptions, not those issued from the negation of the conjecture. In other words, let the input set of clauses S be partitioned into $T \uplus iSOS$, where T contains the clauses issued from the assumptions, and $iSOS$, which stands for *input set of support*, contains those issued from the negation of the conjecture. The hypothesis is that I satisfies T and does not satisfy $iSOS$. Then, two ground clauses are deemed *connected*, if they have complementary literals. The set of *goal-relevant* clauses is defined as the closure of the set of ground instances of clauses in $iSOS$ with respect to connection and resolution. Accordingly, a clause sequence is goal-relevant if all ground instances of all its clauses are. We proved that SGGS is goal sensitive in the sense that it only generates goal-relevant clause sequences. The proof shows that SGGS uses assignments of I-true literals to I-false literals to track literals and ensure that only goal-relevant clause sequences are generated.

Discussion

Perspectives on instance-based theorem proving were given in [23, 25]. They distinguish between methods where the interleaving of instance generation and ground reasoning is *fine-grained*, as in the model-evolution calculus [4, 5, 7, 8, 6], and methods where it is *coarse-grained*, as in Inst-Gen [20, 21, 27, 26], to facilitate the integration of SAT or SMT-solvers, taken "off the shelf." SGGS does not feature either kind of interleaving, and therefore differs from all the methods that do.

The model-evolution calculus generalizes DPLL to first order, with case analysis of disjunction and backtracking: SGGS differs from the model-evolution calculus because it is proof confluent, goal sensitive, and does not reduce to DPLL if given a propositional problem. The Inst-Gen method may need to keep instances of clauses (e.g., storing both C and $C\vartheta$). The same remark applies to methods based on instance generation by clause linking, such as the already mentioned disconnection calculus [10, 29, 30, 31] and hyperlinking strategies [16, 17, 34]. This is a subtle point that is rarely discussed. Informally, in these methods variables represent particular objects, and are not universally quantified like in resolution. In practice, this also means that these methods are not compatible with unrestricted subsumption. On the other hand, in SGGS variables in clauses are universally quantified like in resolution, and in this sense SGGS is fully a first-order method.

For goal-sensitivity, SGGS is reminiscent of *resolution with set of support* [37] for the notion of viewing the input set of clauses as the partition $T \uplus iSOS$, but the two methods differ in many other aspects. For instance, resolution with set of support is not semantically guided, because it assumes that T is consistent, but does not use an initial interpretation I that satisfies T and not $iSOS$. In SGGS the input set of support is used only to define and prove goal-sensitivity, but does not take part in the operations of the method. A feature that sets SGGS aside from all previous methods is the use of assignments between literals to ensure that all derived clause sequences are goal-relevant. Finding a suitable I is a requirement to be met, in order to make SGGS goal-sensitive in practice. However, goal sensitivity is especially relevant when reasoning in large knowledge bases, where an intended model may be known and assumed as initial interpretation.

In this paper we illustrated SGGS in a narrative, informal way. A presentation in terms of inference rules transforming a given SGGS clause sequence into another SGGS clause sequence, under suitable side conditions, is given in [15]. Since SGGS is *model-based à la DPLL*, we plan to investigate also presenting it as an *abstract transition system*, in the style of [33, 12, 19], showing explicitly in the state also the current partial model.

Research on SGGS has just begun, and there are many directions for future work, first and foremost practical inference control and implementation. Implementing SGGS will require to study how to give non-trivial (i.e., not based on sign) initial interpretations, and how to compute efficiently new mechanisms such as most general falsifiers, clause splittings, disposability tests. In the method as described here, the initial interpretation I is like a "default" interpretation, and SGGS generates clause sequences that represent partial models different from I. Another lead could be to make I also *dynamic*, as a parameter that can be varied heuristically during the derivation. Heuristics in DPLL-based SAT-solvers might be a source of inspiration. Other topics for further investigation include the study of SGGS as a model-building method, and its extension to theory reasoning, beginning with equality. Theorem proving has made giant strides, and state of the art systems are very sophisticated in working with mostly syntactic information (e.g., [36, 28, 35]). The challenge of SGGS is to go towards a semantically-oriented style of theorem proving, 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] Peter Baumgartner. Hyper tableaux - the next generation. In Harrie de Swart, editor, *Proceedings of the Twelfth International Conference on Analytic Tableaux and Related Methods (TABLEAUX)*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 60–76. Springer, 1998.

[3] Peter Baumgartner. FDPLL - A first-order Davis-Putnam-Logeman-Loveland procedure. In David McAllester, editor, *Proceedings of the Seventeenth International Conference on Automated Deduction (CADE)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 200–219. Springer, 2000.

[4] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the model evolution calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.

[5] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Lemma learning in the model evolution calculus. In Miki Hermann and Andrei Voronkov, editors, *Proceedings of the Thirteenth International Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 572–586. Springer, 2006.

[6] Peter Baumgartner, Björn Pelzer, and Cesare Tinelli. Model evolution calculus with equality – revised and implemented. *Journal of Symbolic Computation*, 47(9):1011–1045, 2012.

[7] Peter Baumgartner and Cesare Tinelli. The model evolution calculus as a first-order DPLL method. *Artificial Intelligence*, 172(4–5):591–632, 2008.

[8] Peter Baumgartner and Uwe Waldmann. Superposition and model evolution combined. In Renate Schmidt, editor, *Proceedings of the Twenty-Second International Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 17–34. Springer, 2009.

[9] 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 Twenty-Fourth International Conference on Automated Deduction (CADE)*, volume 7898 of *Lecture Notes in Artificial Intelligence*, pages 126–134. Springer, 2013.

[10] Jean-Paul Billon. The disconnection method. In Pierangelo Miglioli, Ugo Moscato, Daniele Mundici, and Mario Ornaghi, editors, *Proceedings of the Tenth International Conference on Analytic Tableaux and Related Methods (TABLEAUX)*, volume 1071 of *Lecture Notes in Artificial Intelligence*, pages 110–126. Springer, 1996.

[11] Maria Paola Bonacina. Towards a unified model of search in theorem proving: subgoal-reduction strategies. *Journal of Symbolic Computation*, 39(2):209–255, 2005.

[12] 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.

[13] Maria Paola Bonacina and David A. Plaisted. Constraint manipulation in SGGS. In Temur Kutsia and Christophe Ringeissen, editors, *Notes of the Twenty-Eighth Workshop on Unification (UNIF), Seventh International Joint Conference on Automated Reasoning (IJCAR) and Sixth Federated Logic Conference (FLoC)*, EasyChair Proceedings in Computing (EPiC), pages 1–8, July 2014.

[14] Maria Paola Bonacina and David A. Plaisted. Model representation by SGGS clause sequences. Submitted for publication (24 pages), 2014.

[15] Maria Paola Bonacina and David A. Plaisted. Semantically-guided goal-sensitive theorem proving. Technical Report 92/2014, Dipartimento di Informatica, Università degli Studi di Verona, January 2014. Revised April 2014, available at `http://profs.sci.univr.it/~bonacina/pub_tr.html` (56 pages).

[16] Heng Chu and David A. Plaisted. Model finding in semantically guided instance-based theorem proving. *Fundamenta Informaticae*, 21(3):221–235, 1994.

[17] Heng Chu and David A. Plaisted. CLINS-S: a semantically guided first-order theorem prover. *Journal of Automated Reasoning*, 18(2), 1997.

[18] Leonardo de Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.

[19] Leonardo de Moura and Dejan Jovanović. A model-constructing satisfiability calculus. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *Proceedings of the Fourteenth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI)*, volume 7737 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2013.

[20] Harald Ganzinger and Konstantin Korovin. New directions in instantiation-based theorem proving. In *Proceedings of the Nineteenth IEEE Symposium on Logic in Computer Science (LICS)*, pages 55–64. IEEE Computer Society Press, 2003.

[21] Harald Ganzinger and Konstantin Korovin. Theory instantiation. In Miki Hermann and Andrei Voronkov, editors, *Proceedings of the Thirteenth International Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 497–511. Springer, 2006.

[22] 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.

[23] Swen Jacobs and Uwe Waldmann. Comparing instance generation methods for automated reasoning. *Journal of Automated Reasoning*, 38:57–78, 2007.

[24] 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 Thirteenth International Conference on Formal Methods in Computer Aided Design (FMCAD)*. ACM and IEEE, 2013.

[25] Konstantin Korovin. An invitation to instantiation-based reasoning: From theory to practice. In Renate Schmidt, editor, *Proceedings of the Twenty-Second International Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 163–166. Springer, 2009.

[26] 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.

[27] Konstantin Korovin and Christoph Sticksel. iProver-Eq: An instantiation-based theorem prover with equality. In Jürgen Giesl and Reiner Hähnle, editors, *Proceedings of the Fifth International Joint Conference on Automated Reasoning (IJCAR)*, volume 6173 of *Lecture Notes in Artificial Intelligence*, pages 196–202. Springer, 2010.

[28] Laura Kovàcs and Andrei Voronkov. First order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Proceedings of the Twenty-Fifth International Conference on Computer-Aided Verification (CAV)*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013.

[29] Reinhold Letz and Gernot Stenz. DCTP - a disconnection calculus theorem prover. In Rajeev P. Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 381–385. Springer, 2001.

[30] Reinhold Letz and Gernot Stenz. Proof and model generation with disconnection tableaux. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Proceedings of the Eighth International Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 142–156. Springer, 2001.

[31] Reinhold Letz and Gernot Stenz. Integration of equality reasoning into the disconnection calculus. In Uwe Egly and Christian G. Fermüller, editors, *Proceedings of the Fifteenth International Conference on Analytic Tableaux and Related Methods (TABLEAUX)*, volume 2381 of *Lecture Notes*

in Artificial Intelligence, pages 176–190. Springer, 2002.

[32] Sharad Malik and Lintao Zhang. Boolean satisfiability from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76–82, 2009.

[33] 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.

[34] David A. Plaisted and Yunshan Zhu. Ordered semantic hyper linking. *Journal of Automated Reasoning*, 25:167–217, 2000.

[35] Stephan Schulz. System description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proceedings of the Nineteenth International Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 8312 of *Lecture Notes in Artificial Intelligence*, pages 735–743. Springer, 2013.

[36] Christoph Weidenbach, Dylana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In Renate Schmidt, editor, *Proceedings of the Twenty-Second International Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 140–145. Springer, 2009.

[37] 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.