Journal of Automated Reasoning manuscript No.
DOI 10.1007/s10817-016-9384-2 in 59(2):165–218, August 2017

Semantically-Guided Goal-Sensitive Reasoning: Inference System and Completeness

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

Received: 9 November 2015 / Accepted: 22 July 2016 / Published online: 6 August 2016

Abstract We present a new method for clausal theorem proving, named SGGS from *semantically-guided goal-sensitive* reasoning. SGGS generalizes to first-order logic the *Conflict-Driven Clause Learning* (CDCL) procedure for propositional satisfiability. Starting from an *initial interpretation*, used for *semantic guidance*, SGGS employs a *sequence of constrained clauses* to represent a candidate model, *instance generation* to extend it, *resolution* and other inferences to *explain* and *solve* conflicts, amending the model. We prove that SGGS is *refutationally complete* and *model complete* in the limit, regardless of initial interpretation. SGGS is also *goal sensitive*, if the initial interpretation is properly chosen, and *proof confluent*, because it repairs the current model without undoing steps by backtracking. Thus, SGGS is a complete first-order method that is *simultaneously* model-based *à la* CDCL, semantically-guided, goal-sensitive, and proof confluent.

Keywords Theorem Proving · Conflict-Driven Clause Learning · Semantic Guidance · Refutational Completeness · Goal Sensitivity

1 Introduction

Theorem proving in first-order logic is a main topic in automated reasoning, and historically could even be considered one of the intellectual birth places of computer science, since the quest for a formulation and a solution of the *Entscheidungsproblem*, or whether there exists an algorithm to decide theoremhood in first-order logic, led Alan M. Turing to invent Turing machines [24]. By the negative answer to the *Entscheidungsproblem*, theorem proving in first-order logic is not decidable; however, it is

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

semi-decidable by *Herbrand's theorem*, which says that a set S of first-order clauses is unsatisfiable if and only if there exists a finite set of ground instances of clauses of S that is unsatisfiable [19]. If S is obtained by transforming into clausal form a set of assumptions and the negation of a conjecture, unsatisfiability means that the conjecture follows from the assumptions. First-order clausal theorem-proving methods implement more or less directly Herbrand's theorem, and the expressive power of the logic makes them desirable, even if they are only semi-decision procedures. Methods for first-order logic may also yield decision procedures for decidable fragments.

In order to be a semi-decision procedure, a theorem-proving method must be *refutationally complete*, meaning that it terminates with a refutation whenever the input set of clauses is unsatisfiable. Although in experiments it is legitimate, and often useful, or even necessary, to use incomplete strategies, establishing refutational completeness is a fundamental way of understanding a method. In this article we present and prove refutationally complete *a new method for first-order theorem proving*, named SGGS, from *Semantically-Guided Goal-Sensitive* reasoning. Our motivation for introducing a new approach, when many exist, is to have a reasoning procedure that is *model-based* and *lifts to first-order logic* the *conflict-driven clause learning* (CDCL) procedure at the heart of contemporary SAT solvers [49,51,48,46].

A reasoning method is *model-based*, if the state of a derivation contains a representation of a candidate (partial) model, and inference and search for a model are intertwined, as inferences build and transform the model while the model drives the inferences. The CDCL procedure represents a partial model of a set of propositional clauses by a sequence of literals, called a *trail*, meaning that literals in the trail are true in the model. Its main operations include *decision*, *clausal propagation*, *explanation* of conflicts, *learning*, and conflict solution by *backjumping*. Decision and backjumping are search operations, while clausal propagation, explanation, and learning are inferences. Decision guesses that a literal is true by adding it to the trail. If a literal is in the trail, all occurrences of its complement are false. Thus, clausal propagation detects *implications* and *conflicts*: if all literals of a clause C but one, say Q, are false, Q is an *implied literal* to be added to the trail with C as *justification*; if all literals of C are false, C is a *conflict clause*. Explanation applies resolution to the conflict clause and a justification, resolving upon the implied literal and its complement in the conflict clause, so that the resolvent is still in conflict. Learning adds a resolvent to the set of clauses. The learned clause is used to backjump to a state of the trail where one of its literals is implied, so that the learned clause is satisfied and becomes a justification.

In order to lift these features to first-order logic, SGGS adopts *semantic guidance* and introduces *uniform falsity*. A reasoning method is *semantically guided*, if its search is driven by a given *fixed* interpretation. Early examples are *semantic resolution* [60], *hyperresolution* [58], and *resolution with set of support* [63]. The reason for *semantic guidance* in SGGS is that while in propositional logic there are finitely many atoms and interpretations, and therefore it makes sense to start the search by guessing truth values, in first-order logic variables in clauses are implicitly universally quantified, atoms have infinitely many ground instances, and there are infinitely many interpretations, so that mere guesses are too uninformed. Thus, SGGS assumes an *initial interpretation*, denoted I, which could be based on sign as in hyperresolu-

tion, or be a model of a subset of input clauses as in resolution with set of support. If I satisfies S, the search is over. Otherwise, SGGS seeks to build a model of S, distinct from I. *Uniform falsity* is crucial for *first-order clausal propagation*, because it restores the symmetry between true and false. In propositional logic, if L is true, $\neg L$ is false, and if L is false, $\neg L$ is true. In first-order logic, a literal is true if all its ground instances are, and it is false if at least one ground instance is: if L is true, all ground instances of $\neg L$ are false, but if L is false, only one ground instance of $\neg L$ is bound to be true. We say that a literal is *uniformly false*, if *all* its ground instances are. Thus, if L is true, $\neg L$ is uniformly false, and if L is uniformly false, $\neg L$ is true.

SGGS represents a partial model of a set S of first-order clauses by a sequence Γ of (possibly constrained) clauses with *selected literals*, called *SGGS clause sequence*. Every literal in Γ must be either I-*true* (true in I) or I-*false* (uniformly false in I), and I-false literals are preferred for selection, to get an interpretation that differs from I where needed to satisfy S. The partial model $I^p(\Gamma)$ represented by Γ is obtained by reading Γ from left to right, and taking for each clause C the ground instances of its selected literal needed to satisfy C, while keeping $I^p(\Gamma)$ consistent. $I^p(\Gamma)$ is completed in an interpretation $I[\Gamma]$ by recurring to I whenever $I^p(\Gamma)$ leaves a ground atom undefined, so that I acts as *default interpretation*. An *SGGS-derivation* is a series $\Gamma_0 \vdash \Gamma_1 \vdash \Gamma_2 \vdash \ldots$ of SGGS clause sequences, where Γ_0 is empty, each subsequent Γ_i is generated from Γ_{i-1} by an SGGS inference, and Γ_i replaces Γ_{i-1}, so that only one SGGS clause sequence exists at any time.

SGGS clause sequences are built primarily by an *instance generation* mechanism, called *SGGS-extension*, that adds to the sequence an instance of an input clause and selects one of its literals. SGGS-extension unifies one or more literals of an input clause with as many selected literals of opposite sign in Γ. Since selected literals define $I[\Gamma]$, SGGS-extension is *model-driven*, and selection of a literal in a clause added by SGGS-extension is the analogue of a CDCL decision. A typical way to start a derivation is to add a clause whose literals are all I-false, so that semantic guidance gives a starting point. Symmetric to SGGS-extension is *SGGS-deletion*, that deletes from Γ any clause satisfied by the partial model induced by clauses of smaller index. Similar to CDCL, a satisfied clause is a clause that the system is done with.

If literal L is selected in Γ, and all ground instances of literal M appear with opposite sign among those that L contributes to $I[\Gamma]$, literal L makes M uniformly false in $I[\Gamma]$. Thus, *first-order clausal propagation* in SGGS works as follows: if all literals of a clause C but one, say Q, are uniformly false in $I[\Gamma]$, Q is an *implied literal* and C is its *justification*; if all literals of C are uniformly false in $I[\Gamma]$, C is a *conflict clause*. Because $I^p(\Gamma)$ is extracted from Γ in left-to-right order, the position of clauses in Γ, like that of literals in the trail of CDCL, is relevant. For example, all justifications are in the *disjoint prefix* of Γ, denoted $dp(\Gamma)$, and defined as the longest prefix of Γ where every selected literal contributes to $I[\Gamma]$ *all* its ground instances. Input unit clauses may enter $dp(\Gamma)$ right away, similarly to the level-0 propagations of CDCL. If SGGS-extension adds to Γ a conflict clause, SGGS *explains the conflict* by a restricted form of first-order resolution, called *SGGS-resolution*, that resolves a conflict clause and a justification, resolving upon the implied literal and a literal of the conflict clause made uniformly false by the implied literal. Similar to CDCL, the resolvent is still in conflict, and if the empty clause arises in this process, the input set

is unsatisfiable. An SGGS-resolvent C is *learned* when it enters $dp(\Gamma)$ by moving left of the clause whose selected literal caused C's selected literal to be uniformly false: this *SGGS-move*, that corresponds to backjumping, solves the conflict by *flipping* the truth value of *all* ground instances of C's selected literal.

A clause in an SGGS clause sequence may be constrained, so that it only represents its ground instances that satisfy the constraint. *SGGS constraints* are a variant of *Herbrand constraints*, that are constraints interpreted in the Herbrand universe [44, 45, 23, 22]. SGGS constraints are introduced by *splitting inference rules*, that replace a clause in Γ by a sequence of *constrained clauses*, such that the union of their sets of ground instances is equal to that of the original clause, but their selected literals are *disjoint*, that is, their atoms do not have ground instances in common. The purpose is to enable SGGS-deletion and SGGS-resolution to get rid of duplications or contradictions between selected literals. Thus, SGGS-splitting is different from splitting in the Davis-Putnam-Logemann-Loveland (DPLL) procedure [26, 25, 19]: splitting in DPLL recalls the branching of a tree (e.g., L on one branch, $\neg L$ on the other), splitting in SGGS recalls the splintering of a piece of wood.

Several theorem-proving strategies are *goal-sensitive* and *proof confluent*. Goal-sensitivity ensures that all generated clauses are connected to the theorem to be proved, which is relevant in practice, especially in case of large axiom sets or knowledge bases. SGGS is goal-sensitive, if I satisfies the clauses issued from the assumptions, not those issued from the theorem. *Proof confluence* means that committing to an inference does not prevent the procedure from finding a proof, so that backtracking, in the sense of undoing steps and returning to a previous state, is not needed. SGGS is proof confluent, because it gets out of a dead end simply by moving a clause in its model representation structure.

This article is organized as follows. Section 2 summarizes the SGGS approach to *model representation* of [18], so that this article is self-contained. Section 3 introduces *SGGS-extension* and *SGGS-splitting* as inference schemes, instantiated into inference rules in Section 4, to form the SGGS inference system, together with *SGGS-deletion*, *SGGS-resolution*, and *SGGS-move*. Section 5 proves two theorems that guarantee that SGGS makes progress: the main one is a *lifting theorem* for SGGS-extension. Section 6 describes *conflict explanation and solving* in SGGS, while Section 7 gives ancillary inference rules for SGGS constraints, that appeared in preliminary form in [16]. After defining *fairness* and *limit* of an SGGS-derivation, Section 8 establishes *refutational completeness* (if S is unsatisfiable, any fair SGGS-derivation from S is a refutation), *model completeness* (if S is satisfiable, any fair SGGS-derivation from S yields a model of S in the limit), and *goal-sensitivity*. The article is closed by a discussion of related work (Section 9.1) and a summary of contributions (Section 9.2), that the reader may want to read as overview of SGGS prior to reading the technical sections. A preliminary exposition of SGGS appeared in [17].

2 First-order Model Representation and Clausal Propagation in SGGS

We assume the standard basic definitions in first-order clausal theorem proving. A clause C is a disjunction of literals, with all variables implicitly universally quantified;

it is treated as a set whenever needed; and $\neg C$ is the implicit universal closure of the negation of C. We write $vars(L)$ for the set of variables occuring in a literal L, and this notation extends to clauses. The symbol $\setminus$ is set subtraction; $at(L)$ is the atom of literal L; and if T is a set of literals, $at(T) = \{at(L) : L \in T\}$. The *size* $|L|$ of an atom L is the number of occurrences of constant, function, and predicate symbols in L (e.g., $|P(f(a),g(a))| = 5$); the size of a literal is that of its atom. Substitutions are denoted by lower case Greek letters and applied in postfix notation, so that $t\vartheta$ is the *instance* of term t obtained by applying substitution ϑ; it is a *ground instance* if ϑ replaces all variables in t by ground terms; it is a *variant* of t if ϑ is a variable renaming, or a substitution that replaces distinct variables by distinct variables. The same applies to literals, clauses, and other expressions. Clause C subsumes clause D, written $C \preceq D$, where $\preceq$ is called *subsumption ordering*, if $C\vartheta \subseteq D$ (as multisets) for some substitution ϑ; then $C \prec D$, if $C \preceq D$ and $D \not\preceq C$. Given a set S of clauses, the *Herbrand universe* is the set of ground terms made of symbols from S, with a new constant symbol added if S has none. *Herbrand interpretations* have the Herbrand universe as domain, and interpret ground terms as themselves. Since they suffice for clausal theorem proving, we consider only Herbrand interpretations, and $\models C$ means $J \models C$ for all Herbrand interpretations J.

SGGS constraints use the symmetric symbol $\equiv$ for identity and the notation $top(t)$ for the top symbol of term t. Then, $\models s \equiv t$, for ground terms s and t, if s and t are the same term; and $\models top(t) = f$, if the top symbol of ground term t is f. From now on we simply write constraint for SGGS constraint. An *atomic constraint* is either $true$ or $false$ or an expression of the form $s \equiv t$ or $top(t) = f$, where s and t are terms, and f is a function symbol. A *constraint* is either an atomic constraint, or the negation, conjunction, or disjunction of constraints. Thus, if $A\vartheta$ is a ground instance of a constraint A, either $\models A\vartheta$ or $\models \neg A\vartheta$. 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 variable symbols. Constraints $top(x) \neq f$ and $x \not\equiv y$ prevent x from being replaced by a term whose top symbol is f, and x and y from being replaced by the same term, respectively.

A *constrained clause* is a formula $A \rhd C$, read "C under A" or "C where 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. The notation $A \rhd C[L]$ says that literal L is *selected* in $A \rhd C$, and $A \rhd L$ is called a *constrained literal*. By convention, 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 not relevant, and $true \rhd C$ is usually abbreviated as C.

Given a constrained clause $A \rhd C$, the set of its *constrained ground instances* (cgi) is $Gr(A \rhd C) = \{C\vartheta : \models A\vartheta,\ C\vartheta\ ground\}$, that is, the set of ground instances that satisfy the constraint ($\models A\vartheta$ means that the existential closure of $A\vartheta$ is satisfied if $A\vartheta$ is not ground). For example, $P(a,b) \in Gr(x \not\equiv y \rhd P(x,y))$, but $P(b,b) \notin Gr(x \not\equiv y \rhd P(x,y))$. If L is the selected literal of C, then $Gr(A \rhd L) = \{L\vartheta : C\vartheta \in Gr(A \rhd C)\}$, and $\neg Gr(A \rhd L) = \{\neg L\vartheta : L\vartheta \in Gr(A \rhd L)\}$. Note how $Gr(false \rhd C) = \emptyset$, while $Gr(true \rhd C)$ contains all ground instances of C. Constrained clauses $A \rhd C$ and $B \rhd D$ are *equivalent* if $Gr(A \rhd C) = Gr(B \rhd D)$.

Subsumption can be generalized to constrained clauses as follows: $A \rhd C \preceq B \rhd D$, if $C\vartheta \subseteq D$ (as multisets) and $B \models A\vartheta$ for some substitution ϑ. If $A \rhd C \preceq B \rhd D$, then

for all $D' \in Gr(B \rhd D)$, there is a $C' \in Gr(A \rhd C)$ such that $C' \subseteq D'$ (as multisets). Indeed, for $D' = C\vartheta\sigma \vee E\sigma$, where E is a disjunction of literals, and $\models B\sigma$ holds, take $C' = C\vartheta\sigma$, which is in $Gr(A \rhd C)$, because $B \models A\vartheta$ and $\models B\sigma$ imply $\models A\vartheta\sigma$.

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. Note that literals of different sign may intersect, as intersection is defined based on atoms. For example, $x \not\equiv y \rhd P(x,y)$ and $top(z) \neq f \rhd P(z,z)$ are disjoint; while the intersection of $P(a,x)$ and $\neg P(z,z)$ is $P(a,a)$. If $A \rhd L$ and $B \rhd M$ do not share variables, as it happens if they are literals from distinct clauses, 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 most general unifier (mgu) of $at(L)$ and $at(M)$. The intersection is given by $at(Gr((A \wedge B)\sigma \rhd M\sigma))$.

For an interpretation J, 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 $J \models Gr(A \rhd C)$.

In propositional logic, if a literal L is false in an interpretation J, its complement $\neg L$ is true in J. If all literals of a clause C are false in J, C is a *conflict clause* with respect to J. In first-order logic, if L is false in J, we only know that some ground instance $L\vartheta$ is false in J and $\neg L\vartheta$ is true. For first-order model-based reasoning, we introduce a stronger notion of falsity, which reproduces the propositional behavior: a literal L is *uniformly false* in J, if *all* its ground instances are false in J, or, equivalently, if $\neg L$ is true in J. In symbols, and for constrained literals, $A \rhd L$ is *uniformly false* in J, if $J \models \neg Gr(A \rhd L)$. A clause C is *uniformly false* in J, if all its literals are, which means that $\neg C$ is true in J. In symbols, and for constrained clauses, $A \rhd C$ is *uniformly false* in J, if $J \models \neg Gr(A \rhd L)$ for all literals L of C. 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, as *all* its constrained ground instances are false in J.

SGGS assumes an *initial interpretation* denoted I. A constrained literal $A \rhd L$ is *I-true*, if $I \models A \rhd L$, and *I-false*, if it is uniformly false in I. Clearly, a constrained literal may be neither I-true nor I-false. A constrained clause $A \rhd C$ is *I-all-true*, if all its literals are I-true, and *I-all-false* if all its literals are I-false. The sets of I-true and I-false literals of C are denoted $tlits(C)$ and $flits(C)$, respectively.

Definition 1 (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; and
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 clause sequence in SGGS corresponds to the trail in CDCL: its purpose is to represent a candidate partial interpretation. Condition (1) is a key invariant of SGGS, while Condition (2) restricts the heuristic choice of selected literal by semantic guidance: in CDCL the choice of decided literal is purely heuristic; in SGGS I-false literals are preferred in order to build an interpretation different from I. The length of Γ, denoted $|\Gamma|$, is given by the number of clauses in Γ. For all j, $1 \leq j \leq |\Gamma|$, clause $A_j \rhd C_j[L_j]$ has *index* j in Γ, and is denoted by $\pi_j[\Gamma]$; the *prefix* of Γ of length j,

denoted $\Gamma|_j$, is given by $\Gamma|_j = A_1 \rhd C_1[L_1], \ldots, A_j \rhd C_j[L_j]$, and is a *proper* prefix if $j < |\Gamma|$. The empty sequence is denoted by ε, so that $\Gamma|_0 = \varepsilon$.

The *partial interpretation induced* by an SGGS clause sequence Γ, denoted $I^p(\Gamma)$, is defined inductively over the length of the sequence: if $|\Gamma| = n$, and $n > 0$, $I^p(\Gamma)$ is 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*. Partial interpretations and proper constrained ground instances are mutually defined as follows:

Definition 2 (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, $I^p(\Gamma) = I^p(\Gamma|_{n-1}) \cup pcgi(A_n \rhd L_n, \Gamma)$.

Definition 3 (Proper constrained ground instances) For $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$, for all j, $1 \leq j \leq |\Gamma|$, the set of *proper constrained ground instances (pcgi)* of clause $A_j \rhd C_j[L_j]$, of its selected literal $A_j \rhd L_j$, and of Γ itself, is defined as follows:

- $pcgi(A_j \rhd C_j[L_j], \Gamma) = \{C[L] \in Gr(A_j \rhd C_j[L_j]) : I^p(\Gamma|_{j-1}) \cap C[L] = \emptyset, \neg L \notin I^p(\Gamma|_{j-1})\}$: $C[L]$ is not satisfied by $I^p(\Gamma|_{j-1})$, as 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; and
- $pcgi(\Gamma) = \bigcup_{j=1}^{n} pcgi(A_j \rhd L_j, \Gamma)$.

It follows that $pcgi(\Gamma) = I^p(\Gamma)$. A constrained ground instance of $A_j \rhd C_j[L_j]$ 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 4 (Complementary constrained ground instances) For $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$, for all j, $1 \leq j \leq |\Gamma|$, 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. If all constrained ground instance of a clause are proper, it means that its selected literal contributes *all* its constrained ground instances to $I^p(\Gamma)$:

Definition 5 (Disjoint prefix) For $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$, the *disjoint prefix* of Γ, denoted $dp(\Gamma)$, is its longest prefix such that, for all j, $1 \leq j \leq |dp(\Gamma)|$, $Gr(A_j \rhd C_j[L_j]) = pcgi(A_j \rhd C_j[L_j], \Gamma)$; if $\Gamma = \varepsilon$, then by convention $dp(\varepsilon) = \varepsilon$.

It follows that a clause in $dp(\Gamma)$ has no complementary constrained ground instances, and no selected literals in $dp(\Gamma)$ intersect, which is the reason for its name: indeed, if $A_i \rhd L_i$ and $A_j \rhd L_j$ intersect, for $i < j$, the constrained ground instances of $A_j \rhd L_j$ whose atoms are in the intersection cannot be proper.

Example 1 Let Γ be the sequence of unit clauses $P(a,x), P(b,y), \neg P(z,z), P(u,v)$, where a and b are constant symbols, and x, y, z, u, and v are variable symbols. All ground instances of $P(a,x)$ and $P(b,y)$ are proper, and none is complementary. All ground instances of $\neg P(z,z)$ are proper, except $\neg P(a,a)$ and $\neg P(b,b)$, that are complementary. Thus, $dp(\Gamma) = P(a,x), P(b,y)$. 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; among these, the complementary ones are those of the form $P(t,t)$ for any ground term t other than a and b.

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. A clause that is already satisfied by the partial interpretation induced by the clauses on its left is *disposable*:

Definition 6 (Disposable clause) A non-empty clause $A \rhd C[L]$ in Γ is *disposable*, if $pcgi(A \rhd L, \Gamma) = ccgi(A \rhd L, \Gamma) = \emptyset$.

Clearly, no clause in $dp(\Gamma)$ can be disposable.

Example 2 Consider for simplicity sequences consisting only of unit clauses whose sole literal is selected. In $P(x)$, $\neg Q(x)$, $P(x)$, the second $P(x)$ is disposable. In $\neg P(x)$, $\neg Q(x)$, $P(x)$, clause $P(x)$ is not disposable, because all its ground instances are complementary. Similarly, in $P(x)$, $\neg P(x)$, $\neg P(x)$, neither occurrence of $\neg P(x)$ is disposable, because for both all ground instances are complementary. If the signature has only one function symbol f and one constant symbol a, then $P(z)$ is disposable in $P(a)$, $P(f(x))$, $P(z)$, while $P(f(x))$ and $P(a)$ are disposable in $P(z)$, $P(a)$, $P(f(x))$.

This example shows a difference between disposability and subsumption: a clause may be made disposable by less general clauses that precede it in Γ, because disposability is a *model-based* notion of redundancy, and the order of occurrence of clauses in Γ affects $I^p(\Gamma)$. Disposability includes subsumption of clauses of higher index:

Lemma 1 *If $\pi_k(\Gamma)$ is disposable, any $\pi_j(\Gamma)$ such that $j > k$ and $\pi_k(\Gamma) \preceq \pi_j(\Gamma)$ is also disposable.*

Proof Let $\pi_k(\Gamma)$ be $A \rhd C$ and $\pi_j(\Gamma)$ be $B \rhd D$. By the disposability hypothesis, we have $I^p(\Gamma|_{k-1}) \models A \rhd C$, or, equivalently, $I^p(\Gamma|_{k-1}) \models Gr(A \rhd C)$. By the subsumption hypothesis, we know that for all $D' \in Gr(B \rhd D)$, there is a $C' \in Gr(A \rhd C)$ such that $C' \subseteq D'$ (as multisets). Thus, $I^p(\Gamma|_{k-1}) \models Gr(B \rhd D)$, hence $I^p(\Gamma|_{k-1}) \models B \rhd D$.

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 7 (Induced interpretation) The *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 other words, $I[\Gamma]$ makes all proper constrained ground instances of all selected literals in Γ true, and otherwise is like I. As a special case, $I[\varepsilon] = I$.

Example 3 If Γ is the sequence $P(a,x), P(b,y), \neg P(z,z), P(u,v)$ of Example 1, and I is the all-negative interpretation, where negative literals are true and positive literals are false, $I[\Gamma] \models P(a,t)$ 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 $I[\Gamma] \models P(s,t)$ for all distinct ground terms s and t.

Every prefix of a sequence is a sequence, and therefore induces an interpretation:

Example 4 Let $\Gamma = C_1, C_2, C_3$ be $[P(x)]$, $\neg P(f(y)) \vee [Q(y)]$, $\neg P(f(z)) \vee \neg Q(g(z)) \vee [R(f(z), g(z))]$, with I all-negative, so that positive literals have been selected by Condition (2) of Definition 1. Then $I[\Gamma|_0] = I[\varepsilon] = I$; $I[\Gamma|_1]$ 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 $R(f(z), g(z))$. Note how for this Γ we have $dp(\Gamma) = \Gamma$.

As proper constrained ground instances of I-true selected literals are true in I, $I[\Gamma]$ and I differ on the proper constrained ground instances of I-false selected literals, that are false in I and true in $I[\Gamma]$. An I-true literal is selected only in an I-all-true clause: if $A_j \rhd C_j[L_j]$ is I-all-true, $I[\Gamma|_{j-1}] = I[\Gamma|_j]$, or I-all-true clauses do not contribute to build the induced interpretation. The rôle of I-all-true clauses will become clear with *first-order clausal propagation*, that we describe next.

Consider 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. Thanks to the notion of uniform falsity, 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, L is *uniformly false* in $I[\Gamma]$; the uniform falsity of L *depends* on the fact that M is selected. SGGS exploits these dependences under *semantic guidance* by I, which is the reason for the requirement that all literals in a sequence be either I-true or I-false (cf. Condition (1) in Definition 1). Under this invariant, if all ground instances of a literal appear negated among ground instances of another, it must be that one is I-true and the other I-false; either an I-true literal depends on an I-false selected literal, or an I-false literal depends on an I-true selected literal:

Definition 8 (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 $L \in flits(C_j[L_j])$.

By Conditions (1) and (2) $A_j \rhd L$ is uniformly false in $I[\Gamma]$. If $A_k \rhd C_k[L_k]$ is in $dp(\Gamma)$, Condition (2) can be rewritten as $\neg Gr(A_j \rhd L) \subseteq Gr(A_k \rhd L_k)$, which holds if there exists a substitution ϑ such that $L = \neg L_k\vartheta$ and $A_j \models A_k\vartheta$. Since $I[\Gamma]$ differs

from Γ for the proper constrained ground instances of I-false selected literals, the most relevant dependences are those where an I-true literal depends on an I-false selected literal: the system modifies I into $I[\Gamma]$ by selecting an I-false literal, but in so doing uniformly falsifies an I-true literal. SGGS uses *assignment functions*, or *assignments* for short, to represent dependences of I-true literals:

Definition 9 (Assignment) Given $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$, for all 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: non-selected I-true literals 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 write that a literal is assigned to a literal or to a clause. Condition (4) is a second restriction on literal selection (after Condition (2) in Definition 1), because it will be fulfilled by selecting in an I-all-true clause a literal that depends on the rightmost clause. With the notion of assignment, we can appreciate a rôle of *I-false selected literals in the disjoint prefix*; since all constrained ground instances of selected literals in $dp(\Gamma)$ are proper, it is easier to assign I-true literals to indices in $dp(\Gamma)$:

Example 5 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 4. The I-true literal $\neg P(f(y))$ in C_2 is assigned to 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 is assigned to $P(x)$. Similarly, the I-true literal $\neg Q(g(z))$ in C_3 is assigned to the selected I-false literal $Q(y)$ in C_2.

Returning to CDCL, if the complements of all the literals of a clause C appear in the trail, C is a *conflict clause*; if the complements of all the literals of C except one, say Q, appear in the trail, Q is an *implied literal* with C as *justification*. By requiring that all I-true literals that are not selected are assigned, and that selected I-true literals are assigned if possible, SGGS ensures that *all I-all-true clauses in Γ are either conflict clauses or justifications of implied literals*. Indeed, let $A_j \rhd C_j[L_j]$ be I-all-true. If all its literals are assigned, it means that they are all uniformly false in $I[\Gamma]$: $A_j \rhd C_j[L_j]$ is *in conflict* with $I[\Gamma]$; all its constrained ground instances are complementary; and it is not disposable. If L_j is not assigned, it means that all other literals of C_j are uniformly false in $I[\Gamma]$, so that satisfying L_j is the only way to satisfy $A_j \rhd C_j[L_j]$: in this sense L_j is an *implied literal* and $A_j \rhd C_j[L_j]$ stands in Γ as its *justification*. We can now appreciate the rôle of *I-true selected literals in the disjoint prefix*: if an I-all-true clause is in $dp(\Gamma)$, its selected literal *cannot* be assigned, because it is disjoint from all selected literals of smaller index. Thus, an

I-true selected literal in $dp(\Gamma)$ and its clause are necessarily an implied literal and its justification. We shall see how SGGS ensures that *all* justifications are in $dp(\Gamma)$.

Also a non-I-all-true $A_j \rhd C_j[L_j]$ can be in conflict. Consider an I-false literal L in $A_j \rhd C_j[L_j]$: if none of its constrained ground instances appear among the proper constrained ground instances of any I-false selected literal L_k with $k < j$, it means that L is uniformly false in $I[\Gamma|_{j-1}]$. If this happens for all I-false literals in $A_j \rhd C_j[L_j]$, as its I-true literals are assigned and therefore uniformly false, $A_j \rhd C_j[L_j]$ is in conflict with $I[\Gamma|_{j-1}]$. How about $I[\Gamma|_j]$? If $A_j \rhd C_j[L_j]$ has proper constrained ground instances, they are satisfied in $I[\Gamma|_j]$, and $A_j \rhd C_j[L_j]$ is not in conflict with $I[\Gamma|_j]$. On the other hand, if all I-false literals in $A_j \rhd C_j[L_j]$ depend on implied literals in $dp(\Gamma)$, no matter which one is selected, this clause has no proper constrained ground instances and is in conflict also with $I[\Gamma|_j]$. We shall see how SGGS *explains* conflicts by resolving I-false literals in conflict clauses with implied literals they depend on: it is the I-false literals in a conflict clause that get resolved away in conflict explanation, because they are those that differentiate $I[\Gamma]$ from I. Unless a contradiction arises, the resulting I-all-true conflict clause will enter $dp(\Gamma)$ and become the justification of its selected literal. It is an I-all-true conflict clause that gets learned, and it is one of its literals that gets implied, because the system has *learned* that all constrained ground instances of this literal ought to be true also in $I[\Gamma]$.

3 The SGGS Inference Schemes

Knowing what SGGS clause sequences represent, in this section and the next we describe how SGGS generates and transforms SGGS clause sequences, hence their induced interpretations. As usual in theorem proving, this process is called *derivation*:

Definition 10 (SGGS-derivation) Given set S of clauses and initial interpretation I, an *SGGS-derivation* from S is a series of triples

$$(S;I;\Gamma_0) \vdash (S;I;\Gamma_1) \vdash \ldots (S;I;\Gamma_j) \vdash \ldots$$

where $\Gamma_0 = \varepsilon$, and $\forall j > 0$, Γ_j is an SGGS clause sequence generated from $(S;I;\Gamma_{j-1})$ by an SGGS-inference.

An SGGS-derivation may be written in the form $\Gamma_0 \vdash \Gamma_1 \vdash \ldots \Gamma_j \vdash \ldots$, as S and I do not change. A derivation is *successful* if there is a k, $k \geq 0$, such that $I[\Gamma_k] \models S$ or Γ_k ($k > 0$) contains the empty clause, denoted by $\bot$; in the latter case the derivation is a *refutation*.

3.1 The SGGS-extension Inference Scheme

SGGS generates an SGGS-derivation seeking a Γ such that $I[\Gamma] \models S$. 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 C' of C such that $I[\Gamma] \not\models C'$. Then, SGGS will apply an *SGGS-extension* inference to generate from C and Γ a possibly constrained clause $A \rhd E$, such that E is an instance of C, and C' is a constrained ground instance of

$A \rhd E$. By adding $A \rhd E$ to Γ, SGGS adds to $I^p(\Gamma)$ the proper constrained ground instances of the selected literal of $A \rhd E$, so that $I[\Gamma]$ satisfies its ground instance that appears in C' (since C' is an instance of E) and therefore C' itself.

Since SGGS-extension generates instances of clauses in S, in order to define it, we need to specify substitutions to instantiate input clauses. SGGS-extension will apply if there are clauses $C \in S$, and $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$ in $dp(\Gamma)$, such that $M_1, \ldots, M_n$ 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, $1 \leq j \leq n$. SGGS-extension unifies literals with I-false selected literals, because they are those that differentiate $I[\Gamma]$ from I; and it picks them in $dp(\Gamma)$, so that all their ground instances are proper, hence relevant to $I[\Gamma]$: in this way instance generation is *model-driven*. This generates the clause $(\bigwedge_{j=1}^{n} B_j\alpha) \rhd C\alpha$. 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. Since every literal in a clause that enters an SGGS clause sequence must be either I-true or I-false, we may need to apply a second substitution β to guarantee that all other literals of the added clause are I-false. If I is based on sign, that is, either *all-negative* (all negative literals are true) or *all-positive* (all positive literals are true), this is *not necessary*: if I is all-negative (all-positive), any input positive (negative) literal is I-false. For I not based on sign, we introduce *falsifiers*, that is, substitutions that capture false ground instances and make literals uniformly false. We say that two sets of substitutions $\mathscr{S}$ and $\mathscr{T}$ are *equivalent* with respect to clause C, written $\mathscr{S} \equiv_C \mathscr{T}$, if they capture the same set of ground instances: $\bigcup_{\vartheta \in \mathscr{S}} Gr(C\vartheta) = \bigcup_{\vartheta \in \mathscr{T}} Gr(C\vartheta)$. Then, a set of substitutions is *J-falsifying* if it captures the ground instances that are false in an interpretation J:

Definition 11 (J-falsifier) Given a clause C and an interpretation J, a set $\mathscr{S}$ of substitutions is *J-falsifying* for C, if $\mathscr{S} \equiv_C \{\vartheta : C\vartheta \in Gr(C) \wedge J \not\models C\vartheta\}$. An element of a J-falsifying set for C is a *J-falsifier* for C.

For the initial interpretation I, we use *semantic falsifying set* and *semantic falsifier* in place of I-falsifying set and I-falsifier. The key point is that if ϑ is a semantic falsifier for C, all literals of $C\vartheta$ are I-false.

As it is typical in first-order inferences, we should not over-commit, and just like we apply a most general unifier, we want to apply a *most general falsifier*. A *most general* substitution is *minimal* with respect to an ordering. For example, a most general unifier is minimal with respect to the ordering on substitutions defined by $\sigma \preceq \vartheta$ if there is a ρ such that $\vartheta = \sigma \circ \rho$, where $\circ$ is composition. In order to define a *most general falsifier* for C, we introduce an ordering on substitutions relative to C: $\vartheta_1 \leq_C^{Gr} \vartheta_2$ if $Gr(C\vartheta_2) \subseteq Gr(C\vartheta_1)$. As usual, $\vartheta_1 <_C^{Gr} \vartheta_2$ if $\vartheta_1 \leq_C^{Gr} \vartheta_2$ and $\vartheta_2 \not\leq_C^{Gr} \vartheta_1$; $\vartheta_1 \equiv_C^{Gr} \vartheta_2$, if $\vartheta_1 \leq_C^{Gr} \vartheta_2$ and $\vartheta_2 \leq_C^{Gr} \vartheta_1$; $\vartheta_1 \geq_C^{Gr} \vartheta_2$ if $\vartheta_2 \leq_C^{Gr} \vartheta_1$; and $\vartheta_1 >_C^{Gr} \vartheta_2$ if $\vartheta_2 <_C^{Gr} \vartheta_1$. Note how to a *smaller* substitution corresponds a *larger* set of ground instances. The following lemma shows that the ordering $<_C^{Gr}$ is *well-founded*; there cannot be an infinite descending chain of substitutions, or, equivalently, there cannot be an infinite ascending chain of ever larger sets of ground instances:

Lemma 2 *For all clauses C, the ordering $<_C^{Gr}$ is well-founded.*

Proof By assumption, the Herbrand universe has at least one constant symbol. If clause C has no function symbols, the theorem is trivial. We assume that C has at least

one function symbol, so that the Herbrand universe is infinite. By way of contradiction, assume that there is an infinite chain $\vartheta_1 >_C^{Gr} \vartheta_2 >_C^{Gr} \vartheta_3 >_C^{Gr} \ldots \vartheta_i >_C^{Gr} \vartheta_{i+1} \ldots$, or $Gr(C\vartheta_1) \subset Gr(C\vartheta_2) \subset Gr(C\vartheta_3) \ldots Gr(C\vartheta_i) \subset Gr(C\vartheta_{i+1}) \ldots$. The strict subset relation implies that no two $C\vartheta_i$'s are equal or variants, that is, there are infinitely many distinct $C\vartheta_i$'s. All $C\vartheta_i$'s have number of literals bounded by the number of literals of C. Since the number of clauses with bounded number of literals and bounded literal size is finite, the sizes of the literals in the $C\vartheta_i$'s must increase without bound. For any n, there is an i such that $C\vartheta_i$ has a literal L of size larger than n, which means that any clause in $Gr(C\vartheta_i)$ has a literal of size larger than n. Suppose the maximum size of any literal in $C\vartheta_1$ is n. Let j be such that $C\vartheta_j$ has a literal of size larger than n. We have $Gr(C\vartheta_1) \subset Gr(C\vartheta_j)$. However, all elements of $Gr(C\vartheta_j)$ have a literal of size larger than n. Because the Herbrand universe has at least one constant symbol, there is a clause $D \in Gr(C\vartheta_1)$ such that the maximum literal size of D is n (D is obtained by replacing all variables by the constant). Therefore $D \notin Gr(C\vartheta_j)$, contradicting $Gr(C\vartheta_1) \subset Gr(C\vartheta_j)$. □

A set $\mathscr{S}$ of substitutions is *most general* for C, if no distinct substitutions ϑ_1 and ϑ_2 in $\mathscr{S}$ satisfy $\vartheta_1 \leq_C^{Gr} \vartheta_2$. In other words, all substitutions in a most general set are incomparable, and therefore, intuitively, independent. By well-foundedness of $<_C^{Gr}$, most general sets exist:

Theorem 1 *For all sets $\mathscr{S}$ of substitutions and clauses C, there is a subset $\mathscr{T} \subseteq \mathscr{S}$ such that $\mathscr{T}$ is most general for C and $\mathscr{S} \equiv_C \mathscr{T}$.*

Proof Since $<_C^{Gr}$ is well-founded, there are minimal elements and we can reason by induction on $<_C^{Gr}$. Let $\mathscr{T}'$ be the subset of the minimal elements in $\mathscr{S}$, that is, those ϑ_2 such that for no $\vartheta_1 \in \mathscr{S}$ it is $\vartheta_1 <_C^{Gr} \vartheta_2$. First we show that $\mathscr{S} \equiv_C \mathscr{T}'$. Since $\mathscr{T}' \subseteq \mathscr{S}$, it is $\bigcup_{\vartheta \in \mathscr{T}'} Gr(C\vartheta) \subseteq \bigcup_{\vartheta \in \mathscr{S}} Gr(C\vartheta)$. For $\bigcup_{\vartheta \in \mathscr{S}} Gr(C\vartheta) \subseteq \bigcup_{\vartheta \in \mathscr{T}'} Gr(C\vartheta)$, we show that for every $\vartheta \in \mathscr{S}$ there is a $\vartheta_2 \in \mathscr{T}'$ such that $\vartheta_2 \leq_C^{Gr} \vartheta$. Indeed, if $\vartheta \in \mathscr{T}'$ the result follows with $\vartheta_2 = \vartheta$. Otherwise, find a $\vartheta_1 \in \mathscr{S}$ such that $\vartheta_1 <_C^{Gr} \vartheta$, and the result follows by induction on ϑ_1. Therefore $\mathscr{S} \equiv_C \mathscr{T}'$. Next, we construct a set $\mathscr{T}$ that is most general for C. By definition of $\mathscr{T}'$, for no $\vartheta_1, \vartheta_2 \in \mathscr{T}'$ can it be that $\vartheta_1 <_C^{Gr} \vartheta_2$. However, there still can be $\vartheta_1, \vartheta_2 \in \mathscr{T}'$ such that $\vartheta_1 \equiv_C^{Gr} \vartheta_2$. Let $\mathscr{T}$ include one element of each $\equiv_C^{Gr}$-equivalence class of $\mathscr{T}'$. Then $\mathscr{T}$ is most general for C and $\mathscr{T} \equiv_C \mathscr{T}'$, so that $\mathscr{S} \equiv_C \mathscr{T}$. □

Corollary 1 *For all interpretations J and clauses C, if there is a J-falsifying set of substitutions for C, there is a most general J-falsifying set of substitutions for C.*

An element of a *most general J-falsifying set* for C is a *most general J-falsifier* for C. For the initial interpretation I, we use *most general semantic falsifying set* and *most general semantic falsifier*. The following theorem gives sufficient conditions to compute most general semantic falsifiers:

Theorem 2 *If I is a Herbrand interpretation such that for any clause C it is decidable whether $I \models C$ and $I \models \neg C$, then most general semantic falsifiers can be computed.*

Proof Assume that we have a clause C and we want to find a most general semantic falsifier for C. If $I \models C$, there is no semantic falsifier. If $I \models \neg C$, then the empty substitution is a most general semantic falsifier. If $I \not\models C$ and $I \not\models \neg C$, let $vars(C) = \{x_1, \ldots, x_n\}$, and let $f_1^{r_1}, \ldots, f_k^{r_k}$ be the function symbols of the signature, each with its arity, including constant symbols as function symbols of arity 0. This yields $n \times k$ substitutions $\vartheta_j^i = \{x_j \leftarrow f_i(y_1, \ldots, y_{r_i})\}$, where $y_1, \ldots, y_{r_i}$ are new variables. Since $I \not\models C$, it is $I \not\models C\vartheta_j^i$ for some ϑ_j^i. Thus, the same procedure applies to each such $C\vartheta_j^i$, until we find an instance $C\vartheta$ such that $I \models \neg C\vartheta$ so that ϑ is a semantic falsifier. Since $I \not\models C$, this procedure is guaranteed to terminate. However, the computed ϑ is not necessarily a most general semantic falsifier. If it is not, given $\vartheta = \{x_1 \leftarrow t_1, \ldots, x_m \leftarrow t_m\}$, a more general semantic falsifier ϑ' can be found by replacing with a new variable one or more subterms of $t_1, \ldots, t_m$, and checking that $I \models \neg C\vartheta'$ still holds. This process terminates and finds eventually a most general semantic falsifier, because the instances $C\vartheta'$ are lower bounded by C itself in the strict subsumption ordering on clauses, which is well-founded. □

We have all the elements to define an *inference scheme for SGGS-extension*:

Definition 12 (SGGS-extension inference scheme) Let S be a set of clauses and Γ an SGGS clause sequence. If there are clauses $C \in S$ and $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$, with $n \geq 0$, in $dp(\Gamma)$, such that

- The literals $M_1, \ldots, M_n$ are I-false,
- 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, $1 \leq j \leq n$, and
- There is a most general semantic falsifier β of $(C \setminus \{L_1, \ldots, L_n\})\alpha$,

the *SGGS-extension inference scheme* generates the clause $A \rhd E = (\bigwedge_{j=1}^n B_j\alpha\beta) \rhd C\alpha\beta$; extends Γ by inserting E at some index larger than the maximum among the indices of $D_1, \ldots, D_n$; and assigns the I-true literals $L_1\alpha\beta, \ldots, L_n\alpha\beta$ of E to $D_1, \ldots, D_n$, respectively. The clauses C, $D_1, \ldots, D_n$, and $A \rhd E$ are called *main premise*, *side premises*, and *extension clause*, respectively.

By construction, all literals of E are either I-true or I-false: $L_1\alpha\beta, \ldots, L_n\alpha\beta$ are I-true, and all the others are I-false. Since $L_j\alpha = \neg M_j\alpha$, for all j, $1 \leq j \leq n$, we have $\neg Gr((\bigwedge_{j=1}^n B_j\alpha\beta) \rhd L_j\alpha\beta) \subseteq Gr(B_j \rhd M_j)$, and since the side premises are in $dp(\Gamma)$, we have $\neg Gr((\bigwedge_{j=1}^n B_j\alpha\beta) \rhd L_j\alpha\beta) \subseteq pcgi(B_j \rhd M_j, \Gamma)$, so that the I-true literals of E *depend* on the selected literals of the side premises and accordingly are *assigned* to them. As a special case, when $n = 0$, there are no side premises, and the conditions involving them are satisfied vacuously. This happens, for instance, at the beginning of the derivation, when SGGS-extension adds to the sequence either I-all-false input clauses (both α and β are empty), or I-all-false instances of input clauses (α is empty and β is not). If I is based on sign, β is empty.

Example 6 Let S be $\{P(a), \neg P(x) \vee Q(f(y)), \neg P(x) \vee \neg Q(z)\}$ with I all-negative. Thus $P(a)$ is an I-all-false input clause, and SGGS-extension adds it to $\Gamma_0 = \varepsilon$ generating $\Gamma_1 = [P(a)]$, where the only literal in the clause is selected: in this step both α and β are empty. Since $I[\Gamma_1]$ satisfies $P(a)$ but no other positive ground literal, all

ground instances of $\neg P(x) \vee Q(f(y))$ of the form $\neg P(a) \vee Q(f(t))$, for t a ground term, are false in $I[\Gamma_1]$. SGGS-extension generates $\Gamma_2 = [P(a)],\ \neg P(a) \vee [Q(f(y))]$, where $Q(f(y))$ is selected because it is I-false: in this step $\alpha = \{x \leftarrow a\}$ is applied to unify the I-true literal $\neg P(x)$ in $\neg P(x) \vee Q(f(y))$ with the I-false selected literal $P(a)$.

As we discussed at the beginning of this section, the purpose of extending Γ with $A \rhd E$ is to modify $I[\Gamma]$ to satisfy more ground instances of input clauses. It may happen, however, that this is not possible, because $A \rhd E$ turns out to be in conflict with $I[\Gamma]$. The generation of a conflict clause alerts the system that Γ cannot be extended, but must be modified to repair $I[\Gamma]$ and solve the conflict. Thus, in Section 4 the SGGS-extension inference scheme will be instantiated into *four SGGS-extension inference rules*, one to add an I-all-true conflict clause, one to add a conflict clause which is not I-all-true, and two to add a non-conflicting clause.

3.2 The SGGS-splitting Inference Scheme

As the clause sequence Γ grows by SGGS-extensions, it may happen that selected literals of its clauses *intersect*. Since proper constrained ground instances of selected literals form $I^p(\Gamma)$, hence $I[\Gamma]$, non-empty intersections of selected literals may represent *duplications*, if the literals have the same sign, or *contradictions*, if the literals have opposite sign. In order to pull out, and then remove, duplications or contradictions, SGGS needs to be able to *partition* a clause. We introduce first a notation that allows us to focus on a literal in a clause even if it is not selected: let $C\langle L\rangle$ denote a clause C containing a literal L, which may be the selected literal but does not have to be. The literal L is said to be *specified* in $C\langle L\rangle$.

Definition 13 (Partition) A *partition* of $A \rhd C\langle L\rangle$, where A is satisfiable, is a set $\{A_i \rhd C_i\langle L_i\rangle\}_{i=1}^n$ such that $Gr(A \rhd C) = \bigcup_{i=1}^n \{Gr(A_i \rhd C_i\langle L_i\rangle)\}$, the constrained literals $A_i \rhd L_i$ are pairwise disjoint, all A_i's are satisfiable, and the L_i's are chosen consistently with L, that is, L_i is an instance of L, for all i, $1 \leq i \leq n$.

This definition requires that A is satisfiable, because otherwise $Gr(A \rhd C)$ would be empty, and an empty set cannot be partitioned.

Example 7 The set $\{true \rhd \langle P(f(z),y)\rangle \vee Q(f(z),y),\ top(x) \neq f \rhd \langle P(x,y)\rangle \vee Q(x,y)\}$ is a partition of $true \rhd \langle P(x,y)\rangle \vee Q(x,y)$. On the other hand, $\{true \rhd P(f(z),y) \vee \langle Q(f(z),y)\rangle,\ top(x) \neq f \rhd P(x,y) \vee \langle Q(x,y)\rangle\}$ is not a partition of $true \rhd \langle P(x,y)\rangle \vee Q(x,y)$, because specified literals are not chosen consistently.

In order to pull out intersections, we need a more specific notion of partition that *isolates* in an element of the partition the constrained ground instances that two literals have in common:

Definition 14 (Splitting) A *splitting* of $A \rhd C\langle L\rangle$ by $B \rhd D[M]$ is a partition $\{A_i \rhd C_i\langle L_i\rangle\}_{i=1}^n$ of $A \rhd C\langle L\rangle$ such that:

1. $\exists j$, $1 \leq j \leq n$, such that $at(Gr(A_j \rhd L_j)) \subseteq at(Gr(B \rhd M))$, and

2. $\forall i$, $1 \leq i \neq j \leq n$, $at(Gr(A_i \rhd L_i))$ and $at(Gr(B \rhd M))$ are disjoint.

$A_j \rhd C_j\langle L_j\rangle$ is the *representative* of $B \rhd D[M]$ in the splitting of $A \rhd C\langle L\rangle$ by $B \rhd D[M]$.

Thus, $at(Gr(A_j \rhd L_j))$ is the intersection of $A \rhd L$ and $B \rhd M$. Computing splittings introduces constraints, including non-standard ones, even when C and D have empty constraints to begin with, and this is why SGGS works with constrained clauses:

Example 8 Given $C = true \rhd P(x,y)$ and $D = true \rhd P(f(w),g(z))$, a splitting of C by D is $\{true \rhd P(f(w),g(z)), top(x) \neq f \rhd P(x,y),\ top(y) \neq g \rhd P(f(x),y)\}$. On the other hand, $\{true \rhd P(f(w),g(z)),\ top(x) \neq f \rhd P(x,y),\ top(y) \neq g \rhd P(x,y)\}$ is not a splitting of C by D, because it is not a partition, since $top(x) \neq f \rhd P(x,y)$ and $top(y) \neq g \rhd P(x,y)$ intersect: for instance, $P(a,b)$ is a constrained ground instance of both. In the correct splitting, $P(a,b)$ is a constrained ground instance of $top(x) \neq f \rhd P(x,y)$, not of $top(y) \neq g \rhd P(f(x),y)$.

A partition is *trivial*, if it is made of only one element, a clause equivalent to the given one. For example, this happens if we try to split a clause by a more general one:

Example 9 Let $A \rhd C[L]$ be $true \rhd P(x,f(x))$ and $B \rhd D[M]$ be $z \not\equiv y \rhd P(z,y)$. Note that $Gr(A \rhd C[L]) \subseteq Gr(B \rhd D[M])$. If we split C by D, the representative of D is given by the intersection of $A \rhd L$ and $B \rhd M$, that is, $A\sigma \wedge B\sigma \rhd C[L]\sigma$, where σ is the mgu of $at(L)$ and $at(M)$. We have $\sigma = \{z \leftarrow x, y \leftarrow f(x)\}$ and the representative is $x \not\equiv f(x) \rhd P(x,f(x))$, which is equivalent to $A \rhd C[L]$ and can be reduced to $A \rhd C[L]$ by constraint manipulation. Thus, the splitting and the partition are trivial.

Definition 15 (SGGS-splitting inference scheme) If Γ is an SGGS clause sequence containing clauses $A \rhd C\langle L\rangle$ and $B \rhd D[M]$, such that L and M intersect, the *SGGS-splitting inference scheme* replaces $A \rhd C\langle L\rangle$ by a splitting of $A \rhd C\langle L\rangle$ by $B \rhd D[M]$ denoted $split(A \rhd C\langle L\rangle, B \rhd D[M])$ or $split(C,D)$ for short.

With a slight abuse of notation, from now on $split(C,D)$ is treated as a sequence, rather than a set as in Definition 14, so that it can replace a clause in a sequence. In the next section the SGGS-splitting inference scheme will be instantiated into three *splitting inference rules*, specifying which of the two clauses is subject to splitting. An application of any splitting inference rule is a *splitting inference*. By isolating intersections in representatives, splitting inferences have two effects. First, they make selected literals disjoint, which grows the disjoint prefix; this is beneficial, because $dp(\Gamma)$ intuitively represents a possibly more stable part of Γ and $I[\Gamma]$. Second, they create situations where all constrained ground instances of a selected literal (e.g., the selected literal L_j of the representative of D in $split(C,D)$) are constrained ground instances of another one (e.g., the selected literal of D). If two such literals have opposite sign, the *SGGS-resolution* inference rule resolves upon them. If they have the same sign, the representative is disposable, and is deleted by *SGGS-deletion*, an inference rule that removes from Γ all disposable clauses. In this way, the inference system amends $I[\Gamma]$ by eliminating contradictions and duplications between selected literals. Thus, all inferences in SGGS are *model-based*: SGGS-extension works with S and Γ to extend Γ and $I[\Gamma]$; all other rules work on Γ to modify $I[\Gamma]$.

4 The SGGS Inference Rules

Several SGGS inference rules employ an ordering on ground literals that also plays a rôle in the completeness proof. The *size ordering* $\prec_s$ on ground atoms is defined by $L \prec_s M$, if $|L| < |M|$, where $<$ is the ordering on the natural numbers. Given two orderings $<$ and $<'$ on a set, $<'$ *extends* $<$ if $<\subseteq<'$, that is, if $a < b$ implies $a <' b$ for all a and b in the set:

Definition 16 (SGGS-suitable ordering) An ordering $\prec$ on ground atoms is *SGGS-suitable* if it is *total* and it extends the size ordering $\prec_s$.

From now on, $\prec$ is an SGGS-suitable ordering on ground atoms. For literals L and M, $L \prec M$ if and only if $at(L) \prec at(M)$. Then, $M \succ L$ if $L \prec M$; $L \preceq M$ if $L \prec M$ or $L = M$; and $M \succeq L$ if $L \preceq M$. An SGGS-suitable ordering is trivially *well-founded*, because $\prec_s$ is well-founded and the number of ground atoms of a given size is finite. Orderings based on size have been traditionally used in theorem proving (e.g., [50]); among more sophisticated orderings, Knuth-Bendix orderings [34,43] correlate well with size and are commonly used. An SGGS-suitable ordering is defined on ground literals because its purpose is to identify minimal elements in sets of constrained ground instances of selected literals. Let M_∞ be a special literal that is larger in $\prec$ than any other literal:

Definition 17 Given a constrained clause $A \rhd C[L]$ occurring in a sequence Γ, the *minimal constrained ground instance* of $A \rhd L$ is

$$cmin(A \rhd L) = \begin{cases} \min_\prec\{M : M \in Gr(A \rhd L)\} & \text{if } Gr(A \rhd L) \neq \emptyset, \\ M_\infty & \text{otherwise;} \end{cases}$$

the *minimal proper constrained ground instance* of $A \rhd L$ is

$$pcmin(A \rhd L, \Gamma) = \begin{cases} \min_\prec\{M : M \in pcgi(A \rhd L, \Gamma)\} & \text{if } pcgi(A \rhd L, \Gamma) \neq \emptyset, \\ M_\infty & \text{otherwise;} \end{cases}$$

and the *minimal complementary constrained ground instance* of $A \rhd L$ is

$$ccmin(A \rhd L, \Gamma) = \begin{cases} \min_\prec\{M : M \in ccgi(A \rhd L, \Gamma)\} & \text{if } ccgi(A \rhd L, \Gamma) \neq \emptyset, \\ M_\infty & \text{otherwise.} \end{cases}$$

In the sequel, SGGS inference rules are presented in the form

$$\frac{\Gamma}{\Gamma'}$$

which means that Γ' is inferred from Γ, and Γ and Γ' are used throughout as the names of the sequences above and below the inference line. Concatenation of sequences is juxtaposition, and the symbol $\square$ is used as a place holder for anonymous subsequences preserved by inferences: for instance, the inference

$$\frac{\square C \square D \square}{\square D \square C \square}$$

only exchanges clauses C and D in the sequence.

4.1 The SGGS-extension Inference Rules

We begin with the inference rules that instantiate the SGGS-extension inference scheme: for the next four inference rules, let $C \in S$ and $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$ in $dp(\Gamma)$, with $n \geq 0$, be the main and side premises that generate extension clause $A \rhd E = (\bigwedge_{j=1}^{n} B_j\alpha\beta) \rhd C\alpha\beta$. Recall that all I-true literals in $A \rhd E$ are assigned to the side premises (cf. Definition 12). The first rule handles the case where the extension clause is I-all-true. Because all its literals are assigned, it is a conflict clause:

Definition 18 (SGGS-extension with I-all-true conflict clause) If the extension clause $A \rhd E$ is I-all-true, the *SGGS-extension inference rule* appends it to Γ:

$$\frac{\Gamma}{\Gamma A \rhd E[L]}$$

and selects in E the literal L assigned to the side premise of largest index.

The selected literal L in $A \rhd E$ is chosen to satisfy Condition (4) in Definition 9. Since L is assigned, $pcgi(A \rhd L, \Gamma') = \emptyset$, and $A \rhd E$ is in conflict with $I[\Gamma'] = I[\Gamma]$.

The second rule covers extension with a conflict clause that is not I-all-true. If all I-false literals in $A \rhd E$ depend on I-true selected literals in $dp(\Gamma)$, $A \rhd E$ is in conflict with $I[\Gamma]$. However, this may hold for instances of $A \rhd E$, rather than for $A \rhd E$ itself, as I-false literals in $A \rhd E$ *intersect* I-true selected literals in $dp(\Gamma)$. Thus, SGGS seeks a most general substitution to capture such instances:

Definition 19 (SGGS-extension with non-I-all-true conflict clause) If the extension clause $A \rhd E$ is not I-all-true, and there exists a most general substitution λ such that for all literals $L \in flits(E)$, there is an I-true selected literal $H \rhd P$ in $dp(\Gamma)$ that $A\lambda \rhd L\lambda$ depends on, the *SGGS-extension inference rule* appends $A\lambda \rhd E[L]\lambda$ to Γ:

$$\frac{\Gamma}{\Gamma A\lambda \rhd E[L]\lambda}$$

where L is an arbitrary I-false literal in $E\lambda$. The substitution λ is called *extension substitution* and also $A\lambda \rhd E[L]\lambda$ is called *extension clause*.

As all I-false literals of $A\lambda \rhd E\lambda$ are uniformly false in $I[\Gamma]$, anyone of them can be selected, so that in this case the inference rule leaves room for a heuristic choice; $A\lambda \rhd E\lambda$ has no proper constrained ground instances (all its constrained ground instances are complementary), and $A\lambda \rhd E\lambda$ is in conflict with $I[\Gamma'] = I[\Gamma]$. An application of SGGS-extension according to Definitions 18 or 19 is called *conflicting*, because the added clause is a conflict clause. A literal is selected also in a conflict clause: the selected literal will be relevant when the conflict is explained and solved.

If the extension clause is *not* in conflict with $I[\Gamma]$, its addition extends Γ into a Γ' such that $I[\Gamma'] \neq I[\Gamma]$. Assume that $A \rhd E$ is not I-all-true, and at least one I-false literal $L \in flits(E)$ *does not intersect* any I-true selected literal: no constrained ground instance of $A \rhd L$ appears negated in $I^p(\Gamma)$, so that if we select L, $A \rhd E[L]$ has no complementary constrained ground instance, and by extending Γ with $A \rhd E[L]$

we would get a Γ' such that $I[\Gamma'] \models A \rhd E[L]$. However, asking that $A \rhd L$ has no complementary constrained ground instance and $I[\Gamma'] \models A \rhd E[L]$ is too much. All we need is to select an $L \in flits(E)$ that has proper constrained ground instances, so that $I[\Gamma'] \models pcgi(A \rhd E[L], \Gamma')$, while $A \rhd E[L]$ may also have complementary constrained ground instances. We begin by negating the condition that characterizes a conflicting extension clause with I-false literals: $A \rhd E$ should have at least one $L \in flits(E)$ that does not depend on any I-true selected literal in $dp(\Gamma)$. However, this is too weak: if all constrained ground instances of L appear among the proper constrained ground instances of an I-false selected literal in Γ, also L will not have proper constrained ground instances. Thus, we require that $A \rhd E$ has an $L \in flits(E)$ such that $at(Gr(A \rhd L)) \not\subseteq at(pcgi(H \rhd P, \Gamma))$ for *all* selected literals $H \rhd P$ in Γ. However, this condition need not apply to the whole Γ. It suffices to find a prefix of Γ such that $A \rhd E$ has proper constrained ground instances if appended to that prefix. We consider first the case where such prefix is Γ itself:

Definition 20 (Non-conflicting non-critical SGGS-extension) If the extension clause $A \rhd E$ is not I-all-true, and for some $L \in flits(E)$, $at(Gr(A \rhd L)) \not\subseteq at(pcgi(H \rhd P, \Gamma))$ for all selected literals $H \rhd P$ in Γ, then any such literal L can be selected, provided $pcgi(A \rhd L, \Gamma A \rhd E[L]) \neq \emptyset$, in which case the *SGGS-extension inference rule* appends $A \rhd E[L]$ to Γ:

$$\frac{\Gamma}{\Gamma A \rhd E[L]}$$

Here too the inference rule leaves room for heuristic choice of selected literal among the I-false ones. We consider next the case where the prefix is proper:

Definition 21 (Critical SGGS-extension) If the extension clause $A \rhd E$ is not I-all-true, and there are a literal $L \in flits(E)$ and a proper prefix Γ^1 of Γ such that $at(Gr(A \rhd L)) \not\subseteq at(pcgi(H \rhd P, \Gamma^1))$ for all selected literals $H \rhd P$ in Γ^1, all side premises are in Γ^1, $\Gamma = \Gamma^1 J \rhd N[O]\Gamma^2$ for a clause $J \rhd N[O]$ such that $pcmin(A \rhd L, \Gamma^1 A \rhd E[L]) \prec pcmin(J \rhd O, \Gamma^1 J \rhd N[O])$, $\Gamma^2 \neq \varepsilon$, and Γ^1 is the shortest such prefix, the *SGGS-extension inference rule* replaces $J \rhd N[O]$ with $A \rhd E[L]$:

$$\frac{\Gamma^1 J \rhd N[O]\Gamma^2}{\Gamma^1 A \rhd E[L]\Gamma^2}$$

In this case a heuristic choice of selected literal applies only if $A \rhd E$ has more than one literal $L \in flits(E)$ satisfying the conditions of the rule for the same shortest prefix Γ^1 and clause $J \rhd N[O]$. An SGGS-extension according to Definitions 20 or 21 is called *non-conflicting*, because the extension clause is *not* in conflict, and it is called *critical* when the extension clause is not appended at the end of the sequence. Since SGGS-extension with non-I-all-true conflict clause may apply only after other inferences placed I-true selected literals in $dp(\Gamma)$, in order to include this kind of SGGS-extension, we allow the next example to feature simple instances of *SGGS-move* and *SGGS-resolution*, whose formal definitions appear in Section 4.3:

Example 10 Let S be $\{\neg P(a) \vee \neg R(y,x),\ P(a),\ \neg P(f(x)) \vee \neg Q(x),\ P(f(x)),\ \neg P(y) \vee R(z, f(z)) \vee Q(y)\}$ with I all-positive. Starting with $\Gamma_0 = \varepsilon$, $I[\Gamma_0] = I \not\models \neg P(a) \vee$

$\neg R(y,x)$: in fact, $\neg P(a) \vee \neg R(y,x)$ is I-all-false. A non-conflicting non-critical SGGS-extension yields $\Gamma_1 = [\neg P(a)] \vee \neg R(y,x)$. In this example we assume a heuristic that selects the leftmost eligible literal whenever there is a choice. Next, $I[\Gamma_1] \not\models P(a)$: $P(a)$ is in conflict with $I[\Gamma_1]$. SGGS-extension with I-all-true conflict clause generates $\Gamma_2 = [\neg P(a)] \vee \neg R(y,x),\ [P(a)]$. To solve the conflict, SGGS moves $P(a)$ left of $\neg P(a)$, so that $\Gamma_3 = [P(a)],\ [\neg P(a)] \vee \neg R(y,x)$, and resolves them, producing $\Gamma_4 = [P(a)],\ [\neg R(y,x)]$. As $I[\Gamma_4] \not\models \neg P(f(x)) \vee \neg Q(x)$, non-conflicting non-critical SGGS-extension yields $\Gamma_5 = [P(a)],\ [\neg R(y,x)],\ [\neg P(f(x))] \vee \neg Q(x)$. Now $P(f(x))$ is in conflict with $I[\Gamma_5]$ and SGGS-extension with I-all-true conflict clause produces $\Gamma_6 = [P(a)],\ [\neg R(y,x)],\ [\neg P(f(x))] \vee \neg Q(x),\ [P(f(x))]$. SGGS-move generates $\Gamma_7 = [P(a)],\ [\neg R(y,x)],\ [P(f(x))],\ [\neg P(f(x))] \vee \neg Q(x)$; and SGGS-resolution yields $\Gamma_8 = [P(a)],\ [\neg R(y,x)],\ [P(f(x))],\ [\neg Q(x)]$. At this stage $I[\Gamma_8] \not\models \neg P(y) \vee R(z,f(z)) \vee Q(y)$. While in all SGGS-extensions so far α is empty, $\alpha = \{y_1 \leftarrow z, x_1 \leftarrow f(z), x_3 \leftarrow y\}$ is the simultaneous mgu of literals $R(z,f(z))$ and $Q(y)$ in $C = \neg P(y) \vee R(z,f(z)) \vee Q(y)$ with $\neg R(y_1,x_1)$ and $\neg Q(x_3)$ in $\Gamma_8 = [P(a)],\ [\neg R(y_1,x_1)],\ [P(f(x_2))],\ [\neg Q(x_3)]$ (after variable renaming), so that the extension clause is $C\alpha = C$. The I-false literal $\neg P(y)$ in C intersects both I-true literals $P(a)$ and $P(f(x))$ in $dp(\Gamma)$. Extension substitution $\lambda_1 = \{y \leftarrow a\}$ yields the conflict clause $\neg P(a) \vee R(z,f(z)) \vee Q(a)$, and extension substitution $\lambda_2 = \{y \leftarrow f(x)\}$ yields the conflict clause $\neg P(f(x)) \vee R(z,f(z)) \vee Q(f(x))$. Thus, two instances of SGGS-extension with non-I-all-true conflict clause may apply.

This example also shows that there may be more than one extension substitution for a given Γ and extension clause.

4.2 Splitting Inference Rules

We consider next the *splitting inference rules* that instantiate the SGGS-splitting inference scheme (cf. Definition 15). These rules employ the following criteria to choose the first clause of a partition viewed as a subsequence:

Definition 22 (Preferred clause in a partition) Whenever an SGGS inference $\Gamma \vdash \Gamma'$ replaces a clause $A \rhd C\langle L\rangle$ by its partition $\{A_i \rhd C_i\langle L_i\rangle\}_{i=1}^n$, the *preferred clause in the partition* is the clause $A_k \rhd C_k\langle L_k\rangle$, for some k, $1 \leq k \leq n$, such that

1. If $pcgi(A \rhd C, \Gamma) \neq \emptyset$, then $pcmin(A_k \rhd L_k, \Gamma') = min_{\prec}\{pcmin(A_i \rhd L_i, \Gamma') : 1 \leq i \leq n\}$; and
2. If $pcgi(A \rhd C, \Gamma) = \emptyset$ and $ccgi(A \rhd C, \Gamma) \neq \emptyset$, then $ccmin(A_k \rhd L_k, \Gamma') = min_{\prec}\{ccmin(A_i \rhd L_i, \Gamma') : 1 \leq i \leq n\}$.

These criteria will make sense with the completeness proof (cf. Section 8). The preferred clause is uniquely defined because $\prec$ is total on ground literals. In the first splitting inference rule a clause splits a clause of larger index:

Definition 23 (Splitting by similar/dissimilar literal) If $A \rhd C[L]$ occurs to the left of $B \rhd D\langle M\rangle$ and in $dp(\Gamma)$, L and M intersect, and

– Either they have the same sign and truth value in I (*similar literal*),

– Or they have opposite sign and truth value in I (*dissimilar literal*), and M is the selected literal in clause D,

the *splitting inference rule* replaces D by $split(D,C) = D_1, \ldots, D_n$:

$$\frac{\Box C \Box D \Box}{\Box C \Box split(D,C) \Box}$$

where D_1 is the preferred clause in the partition, and the I-true literals in $D_1, \ldots, D_n$ are assigned to the same clauses that the corresponding literals in D were assigned to.

Since all literals of all clauses in a sequence are either I-true or I-false, having the same truth value in I means both I-true or both I-false, while having opposite truth value in I means that one is I-true and the other is I-false. Clauses in $split(D,C)$ may be deleted without affecting completeness, even if they are not disposable, provided D_1 is kept. Splitting by similar literal is abbreviated *similar splitting* or *s-splitting*, and splitting by dissimilar literal is abbreviated *dissimilar splitting* or *d-splitting*.

Example 11 Assume that I is all-positive and two SGGS-extension steps create the sequence $\neg P(a,x), \neg P(x,b)$: s-splitting $\neg P(x,b)$ by $\neg P(a,x)$ generates the sequence $\neg P(a,x), \neg P(a,b), x \not\equiv a \rhd \neg P(x,b)$, where $\neg P(a,b)$ is disposable, so that its deletion yields $\neg P(a,x), x \not\equiv a \rhd \neg P(x,b)$.

The next lemma proves that disposability of the representative holds in general for s-splitting:

Lemma 3 *If Γ' is inferred from Γ by s-splitting $B \rhd D\langle M\rangle$ by $A \rhd C[L]$, the representative of $A \rhd C[L]$ in $split(D,C)$ is disposable in Γ'.*

Proof Let $B_j \rhd D_j\langle M_j\rangle$ be the representative of C in $split(D,C)$. By Definition 14, this means that $at(Gr(B_j \rhd M_j)) \subseteq at(Gr(A \rhd L))$. Since s-splitting applied, L and M, hence L and M_j, have the same sign, so that we also have $Gr(B_j \rhd M_j) \subseteq Gr(A \rhd L)$. Since C is in $dp(\Gamma)$, by Definition 5, we have that $Gr(A \rhd C[L]) = pcgi(A \rhd C[L], \Gamma) = pcgi(A \rhd C[L], \Gamma')$, and $Gr(A \rhd L) = pcgi(A \rhd L, \Gamma) = pcgi(A \rhd L, \Gamma')$. Let i and k, with $i < k$, be the indices of C and D_j, respectively, in Γ': we have $Gr(A \rhd L) \subseteq I^p(\Gamma'|_{k-1})$. From this and $Gr(B_j \rhd M_j) \subseteq Gr(A \rhd L)$, it follows that $I^p(\Gamma'|_{k-1})$ satisfies $B_j \rhd D_j\langle M_j\rangle$ which is therefore disposable. □

This proof does not require M to be selected, and this is why s-splitting may split a clause on a literal other than the selected one: if it were restricted to the case where M is selected, we could fail to isolate intersections and remove disposable clauses (cf. also Lemma 7 in Section 5).

The second splitting inference rule is called *left splitting*, because it splits the clause of smaller index: this happens only to cover the intersection between the selected literal M of an I-all-true clause $B \rhd D[M]$ and the I-false literal L of the clause $A \rhd C[L]$ in $dp(\Gamma)$ that M is assigned to. For example, $B \rhd D[M]$ is an I-all-true conflict clause added by SGGS-extension and $A \rhd C[L]$ the rightmost side premise (cf. Definition 18). By definition of assignment, dependence, and disjoint prefix, we have $\neg Gr(B \rhd M) \subseteq pcgi(A \rhd L, \Gamma) = Gr(A \rhd L)$, so that $at(Gr(B \rhd M)) \subseteq$

$at(Gr(A \rhd L))$. The inference rule requires that $\neg Gr(B \rhd M) \subset pcgi(A \rhd L, \Gamma)$, because if $\neg Gr(B \rhd M) = pcgi(A \rhd L, \Gamma)$, then $at(Gr(B \rhd M)) = at(Gr(A \rhd L))$ and there is no point in splitting $A \rhd C[L]$ by $B \rhd D[M]$, as $A \rhd L$ does not have instances outside of the intersection:

Definition 24 (Left splitting) If $A \rhd C[L]$ occurs to the left of $B \rhd D[M]$ and in $dp(\Gamma)$, D is I-all-true, M is assigned to C with $\neg Gr(B \rhd M) \subset pcgi(A \rhd L, \Gamma)$, and there is no other literal of D that is assigned to C and unifies with M, the *left splitting* inference rule replaces C by $split(C, D) = C_1, \ldots, C_n$:

$$\frac{\Box C \Box D \Box}{\Box split(C, D) \Box D \Box}$$

where C_1 is the preferred clause in the partition and M is assigned to the representative of D in $split(C, D)$.

Here too it is permissible to delete clauses in $split(C, D)$ as long as C_1 and the representative of D in $split(C, D)$ are kept. Let $A_j \rhd C_j[L_j]$ be the representative: by combining $at(Gr(B \rhd M)) \subset at(Gr(A \rhd L))$ (cf. the above discussion and Definition 24) with $at(Gr(A_j \rhd L_j)) \subseteq at(Gr(B \rhd M))$ (cf. Condition (1) of Definition 14), we have $at(Gr(A_j \rhd L_j)) = at(Gr(B \rhd M))$, and moreover $\neg Gr(B \rhd M) = pcgi(A_j \rhd L_j, \Gamma)$, because $A \rhd C[L]$ is in $dp(\Gamma)$ by Definition 24 and $A_j \rhd C_j[L_j]$ is in $dp(\Gamma')$ by Condition (2) of Definition 14. The purpose of left splitting is precisely to transform a Γ where the selected literal M of an I-all-true conflict clause $B \rhd D[M]$ is assigned to $A \rhd C[L]$ in $dp(\Gamma)$ with $\neg Gr(B \rhd M) \subset pcgi(A \rhd L, \Gamma)$, into a Γ' where M is assigned to $A_j \rhd C_j[L_j]$ in $dp(\Gamma')$ with $\neg Gr(B \rhd M) = pcgi(A_j \rhd L_j, \Gamma')$. This condition is relevant to the inference rule *SGGS-move*, that we consider next.

4.3 Conflict-Solving Inference Rules: SGGS-move and SGGS-resolution

The purpose of SGGS-move is *to solve* the conflict represented by an I-all-true conflict clause $B \rhd D[M]$ by moving it to the left of the clause $A \rhd C[L]$ in $dp(\Gamma)$ that M is assigned to. This move requires that no literal of D other than M is assigned to C, because if $M' \in D$ were also assigned to C, then M' would have nowhere to be assigned after the move. The non-unifiability condition in the definition of left splitting (cf. Definition 24) ensures precisely that no literal other than M in $B \rhd D[M]$ gets assigned to the representative of $B \rhd D[M]$ in the splitting produced by left splitting, so that after a left splitting inference the requirement for SGGS-move is satisfied. Of course, SGGS-move can be applied also without being preceded by left splitting: if $\neg Gr(B \rhd M) \subset pcgi(A \rhd L, \Gamma)$, SGGS applies first left splitting and then SGGS-move; if $\neg Gr(B \rhd M) = pcgi(A \rhd L, \Gamma)$, SGGS applies SGGS-move directly.

Definition 25 (SGGS-move) If $A \rhd C[L]$ occurs to the left of $B \rhd D[M]$ and in $dp(\Gamma)$, D is I-all-true, M is assigned to C, no other literal of D is assigned to C, and $\neg Gr(B \rhd M) = pcgi(A \rhd L, \Gamma)$, the *SGGS-move* inference rule moves D to the left of C:

$$\frac{\Box C \Box D \Box}{\Box D C \Box \Box}$$

The condition that no literal of D besides M is assigned to C plays a rôle in the proof of the following lemma, which shows the effects of an SGGS-move step:

Lemma 4 *If $\Gamma \vdash \Gamma'$ is an SGGS-move step that moves $B \rhd D[M]$ to the left of $A \rhd C[L]$, $B \rhd D[M]$ is in $dp(\Gamma')$ and $at(pcgi(B \rhd M, \Gamma')) = at(pcgi(A \rhd L, \Gamma))$.*

Proof Let i be the position of C in Γ and of D in Γ', so that $\Gamma|_{i-1} = \Gamma'|_{i-1}$. By definition of SGGS-move, M is assigned to i in Γ and no literal of D is assigned to i. Since the selected literal is assigned rightmost (cf. Condition (4) in Definition 9), all the literals of D other than M are assigned to indices strictly smaller than i. This means that they are uniformly false in $I[\Gamma|_{i-1}] = I[\Gamma'|_{i-1}]$. It follows that for all $D'[M'] \in Gr(B \rhd D[M])$, no literal other than M' can be in $I^p(\Gamma'|_{i-1})$. By definition of SGGS-move, $\neg Gr(B \rhd M) = pcgi(A \rhd L, \Gamma)$, so that for all $M' \in Gr(B \rhd M)$, there is an $L' \in pcgi(A \rhd L, \Gamma)$ such that $M' = \neg L'$. Neither M' nor $\neg M'$ can be in $I^p(\Gamma'|_{i-1})$: if M' were in $I^p(\Gamma'|_{i-1})$, L' would be a ccgi and not a pcgi; if $\neg M'$ were in $I^p(\Gamma'|_{i-1})$, L' would be also, and it would not be a pcgi. Thus, $pcgi(B \rhd M, \Gamma') = Gr(B \rhd M)$, $B \rhd D[M]$ is in $dp(\Gamma')$, and $at(pcgi(B \rhd M, \Gamma')) = at(pcgi(A \rhd L, \Gamma))$. □

The effect of an SGGS-move on the induced interpretation is that where $I[\Gamma]$ contains $pcgi(A \rhd L, \Gamma)$, $I[\Gamma']$ contains instead $pcgi(B \rhd M, \Gamma')$: since $A \rhd L$ and $B \rhd M$ have the same (proper) constrained ground instances with opposite sign, the effect of the move is to *flip* at once the truth value of *all* (possibly infinitely many) ground atoms in $at(Gr(B \rhd M)) = at(Gr(A \rhd L))$. SGGS-move realizes at the first-order level the effect of a *backjump* (which flips the truth value of a propositional literal) in CDCL. The conflict is solved and $B \rhd D[M]$ enters $dp(\Gamma')$ to be the *justification* of its selected literal M that remains unassigned and is an *implied literal*. Since conflicting I-all-true clauses are appended by SGGS-extension at the rightmost end of the sequence and they become justifications by entering the disjoint prefix when the conflict is solved by SGGS-move, it follows that *all* justifications are in the disjoint prefix. SGGS-move is also the analogue of *learning* in CDCL: the system learns $B \rhd D[M]$ by putting it in $dp(\Gamma')$.

An I-all-true clause $B \rhd D[M]$ can resolve with a clause on its right, resolving upon its implied literal M and a selected I-false literal that *depends* on M:

Definition 26 (SGGS-resolution) If $B \rhd D[M]$ occurs to the left of $A \rhd C[L]$ and in $dp(\Gamma)$, $B \rhd D[M]$ is I-all-true, L is I-false, $L = \neg M\vartheta$ for some substitution ϑ, and $A \models B\vartheta$, the *SGGS-resolution* inference rule generates the *SGGS-resolvent* $Res(C,D) = A \rhd R$, where R is $(C \setminus \{L\}) \cup (D \setminus \{M\})\vartheta$, and replaces C by $Res(C,D)$:

$$\frac{\Box D \Box C \Gamma^{\dagger}}{\Box D \Box Res(C,D) \Gamma^{\dagger\dagger}}$$

where $\Gamma^{\dagger\dagger}$ is $\Gamma^{\dagger}$ with all clauses including literals assigned to C deleted, and for all $P\vartheta \in tlits(R)$, $P\vartheta$ is assigned to the same clause that literal $P \in tlits(C) \cup tlits(D)$ was assigned to. An application of this rule is denoted by $res(A \rhd C[L], B \rhd D[M], A \rhd R)$.

Assignments can be inherited, because if P depends on a literal, so does $P\vartheta$. Thus, all I-true literals in an SGGS-resolvent are assigned. If an SGGS-resolvent is I-all-true, its literal assigned rightmost is selected. Otherwise, an I-false literal is selected,

and if an SGGS-resolvent has more than one, a heuristic choice will apply. Since $B \rhd D[M]$ is I-all-true and in $dp(\Gamma)$, $B \rhd D[M]$ is a justification and M an implied literal. The conditions $L = \neg M\vartheta$ and $A \models B\vartheta$ mean that $\neg Gr(A \rhd L) \subseteq Gr(B \rhd M) = pcgi(B \rhd M, \Gamma)$. Thus, SGGS-resolution resolves the I-true implied literal $B \rhd M$ with a selected I-false literal $A \rhd L$ that *depends* on $B \rhd M$. After moving $B \rhd D[M]$ to the disjoint prefix, the system "cleans" the sequence by resolving away literals made uniformly false by the move:

Lemma 5 *If $A \rhd C[L]$ and $B \rhd D[M]$ SGGS-resolve in Γ, then $pcgi(A \rhd L, \Gamma) = \emptyset$.*

Proof It follows from $\neg Gr(A \rhd L) \subseteq pcgi(B \rhd M, \Gamma)$. □

SGGS-resolution is the only SGGS inference that can generate $\bot$. It removes $A \rhd C[L]$ from Γ, because it does not contribute to $I[\Gamma]$, and replaces it by either $\bot$ or an $A \rhd R$ that may contribute proper or complementary constrained ground instances:

Lemma 6 *If $A \rhd C[L]$ and $B \rhd D[M]$ SGGS-resolve, $A \rhd C[L]$ is not disposable, and $Res(C,D) \neq \bot$, then $Res(C,D)$ is not disposable.*

Proof Let $\Gamma \vdash \Gamma'$ be an SGGS-resolution step with premises $A \rhd C[L]$ and $B \rhd D[M]$, where $\Gamma = \Gamma^1 D \Gamma^2 C \Gamma^\dagger$, and $\Gamma' = \Gamma^1 D \Gamma^2 Res(C,D) \Gamma^{\dagger\dagger}$ according to Definition 26. Since $pcgi(A \rhd L, \Gamma) = \emptyset$ by Lemma 5, but $A \rhd C[L]$ is not disposable by hypothesis, it follows that $ccgi(A \rhd L, \Gamma) \neq \emptyset$. Let $C'[L']$ be a ccgi of $A \rhd C[L]$. Since $\neg Gr(A \rhd L) \subseteq Gr(B \rhd M) = pcgi(B \rhd M, \Gamma)$, there exists a $D'[M'] \in Gr(B \rhd D[M])$ such that $\neg L' = M'$. Without loss of generality, let Γ^1 be the longest prefix of Γ not containing D, and $\Gamma'' = \Gamma^1 D \Gamma^2$ the longest prefix of Γ not containing C. Consider the cgi $(C' \setminus \{L'\}) \cup (D' \setminus \{M'\})$ of $Res(C,D)$. We show that this cgi has no intersection with $I^p(\Gamma'')$. For the literals in $C' \setminus \{L'\}$, this follows from the fact that C' is a ccgi of C (cf. Definition 4). For any literal Q' in $D' \setminus \{M'\}$, let Q be the literal in $D \setminus \{M\}$ of which Q' is instance. Since D is I-all-true, Q is I-true; since Q is not M, Q is assigned to some clause $E[P]$ in Γ^1. Thus, Q' appears negated among the pcgi's of P, that are included in $I^p(\Gamma^1)$ and in $I^p(\Gamma'')$. Therefore Q' cannot occur in $I^p(\Gamma'')$, because otherwise $I^p(\Gamma'')$ would be inconsistent. Since at least a cgi of $Res(C,D)$ does not intersect $I^p(\Gamma'')$, $Res(C,D)$ is not disposable in Γ'. □

Left splitting and SGGS-resolution apply to *dual* situations: left splitting applies when an I-true selected literal depends on an I-false selected literal in $dp(\Gamma)$; SGGS-resolution applies when an I-false selected literal depends on an I-true selected literal in $dp(\Gamma)$. When left splitting, SGGS-move, and SGGS-resolution are applied in this order, an I-all-true conflict clause splits the clause that its selected literal is assigned to, moves left of its representative in the splitting, and then resolves with it.

4.4 More Splitting Inference Rules and Bookkeeping Rules

The splitting inference rules also comprise *SGGS-factoring*. This rule applies when left splitting does not (cf. Definition 24), because the I-all-true conflict clause $B \rhd D[M]$ has more than one literal assigned to the clause $A \rhd C[L]$ which M is assigned to. If these literals unify, SGGS-factoring generates a factor of the I-all-true clause, and splits the I-all-true clause by its factor:

Definition 27 (SGGS-factoring) If $A \rhd C[L]$ occurs to the left of $B \rhd D[M]$ and in $dp(\Gamma)$, D is I-all-true, M is assigned to C, and there is another literal Q in D, that is assigned to C, has the same sign as M, and unifies with M with most general unifier ϑ, the *SGGS-factoring* inference rule generates the *factor* $D_f = B\vartheta \rhd D[M]\vartheta$, and replaces D by $split(D, D_f) = D_1, \ldots, D_n$:

$$\frac{\Box C \Box D \Box}{\Box C \Box split(D, D_f) \Box}$$

where D_1 is the preferred clause in the partition, D_f is its own representative in $split(D, D_f)$, and for all $P\vartheta \in tlits(D_f)$, $P\vartheta$ is assigned to the same clause that literal $P \in tlits(D)$ was assigned to.

Note that D_f is also I-all-true and a conflict clause, and it is its own representative in $split(D, D_f)$ by Definition 14. Similar to SGGS-resolution (cf. Definition 26) assignments are inherited. In particular, the selected literal $M\vartheta$ of D_f is assigned to C. As before, clauses in $split(D, D_f)$ can be deleted as a heuristic, except for D_1 and D_f, which must be kept. If $M\vartheta$ still unifies with other literals of D_f, then additional SGGS-factoring steps can be applied to C and D_f. If $M\vartheta$ does not unify with any other literals of D_f, then left splitting can be applied to C and D_f. Thus, the sequence left splitting, SGGS-move, and SGGS-resolution can be preceded by one or more SGGS-factoring steps.

The SGGS inference system also features a few bookkeeping rules. The first one deletes disposable clauses, because they are useless to find a refutation or a different model, since they are satisfied by $I[\Gamma]$ (cf. Definition 6):

Definition 28 (SGGS-deletion) The *SGGS-deletion* inference rule deletes any disposable clause in Γ:

$$\frac{\Gamma}{\Gamma'}$$

where Γ' is Γ with all disposable clauses removed.

SGGS-deletion helps keeping the clause sequence from growing too long. *SGGS-sorting* reorders clauses in $dp(\Gamma)$ so that clauses with small minimal proper constrained ground instance migrate as far as possible to the left:

Definition 29 (SGGS-sorting) If $A \rhd C[L]$ occurs to the left of $B \rhd D[M]$, both are in $dp(\Gamma)$, no literal of D is assigned to C, and $pcmin(B \rhd M, \Gamma) \prec pcmin(A \rhd L, \Gamma)$, the *SGGS-sorting* inference rule reorders them:

$$\frac{\Box CD\Gamma^{\dagger}}{\Box DC\Gamma^{\dagger\dagger}}$$

where $\Gamma^{\dagger\dagger}$ is $\Gamma^{\dagger}$ with $F[Q]$ replaced by $F[P]$ if there is an I-all-true clause F whose literals Q and P are assigned to D and C, respectively.

The latter provision ensures that a selected literal remains assigned rightmost (cf. Condition (4) in Definition 9). SGGS-sorting can be applied multiple consecutive times. It is not needed for completeness, but it might help to generate proofs faster. The last rule, named *recursive partitioning*, uses the notion of *literal mapping*: given sets of literals $\mathscr{M} = \{M_1, \ldots, M_k\}$ and $\mathscr{L} = \{L_1, \ldots, L_n\}$, a *literal mapping* from $\mathscr{M}$ to $\mathscr{L}$ is a function $f\colon \{1, \ldots, k\} \to \{1, \ldots, n\}$.

Definition 30 A literal mapping f from $\mathscr{M} = \{M_1, \ldots, M_k\}$ to $\mathscr{L} = \{L_1, \ldots, L_n\}$ is *compatible*, if there exists a simultaneous most general unifier ϑ_f such that for all i, $1 \leq i \leq k$, $M_i\vartheta_f = \neg L_{f(i)}\vartheta_f$, assuming that for all i, j, $1 \leq i \neq j \leq n$, $vars(L_i) \cap vars(L_j) = \emptyset$.

Recursive partitioning applies after a splitting inference replaces a clause by a partition where the specified literal is the selected one:

Definition 31 (Recursive partitioning) Let $A \rhd C[L]$ be a clause that was replaced by its partition $\Gamma_C = \{A_i \rhd C_i[L_i]\}_{i=1}^n$, $B \rhd D$ a clause with literals assigned to $A \rhd C[L]$, $\mathscr{L} = \{L_1, \ldots, L_n\}$ the set of the selected literals in Γ_C, and $\mathscr{M} = \{M_1, \ldots, M_k\}$ the set of the literals of D that were assigned to C. The *recursive partitioning* inference rule replaces $B \rhd D$ by its partition Γ_D:

$$\frac{\Box\Gamma_C\Box D\Box}{\Box\Gamma_C\Box\Gamma_D\Box}$$

made of all clauses $[(\bigwedge_{i=1}^k A_{f(i)}\vartheta_f) \wedge B\vartheta_f] \rhd D\vartheta_f$ such that f is a compatible literal mapping from $\mathscr{M}$ to $\mathscr{L}$, and the constraint $(\bigwedge_{i=1}^k A_{f(i)}\vartheta_f) \wedge B\vartheta_f$ is satisfiable. For all i, $1 \leq i \leq k$, the literal $M_i\vartheta_f$ in $[(\bigwedge_{i=1}^k A_{f(i)}\vartheta_f) \wedge B\vartheta_f] \rhd D\vartheta_f$ is assigned to $A_{f(i)} \rhd C_{f(i)}[L_{f(i)}]$; and the first clause in Γ_D is the preferred clause in the partition.

The fact that $M \in \mathscr{M}$ was assigned to $A \rhd C[L]$, means that $\neg Gr(B \rhd M) \subseteq pcgi(A \rhd L, \Gamma) \subseteq Gr(A \rhd L)$. When $A \rhd C[L]$ gets partitioned into $\{A_i \rhd C_i[L_i]\}_{i=1}^n$, the sets $Gr(A_i \rhd L_i)$ are subsets of $Gr(A \rhd L)$, and that is why recursive partitioning considers all the ways the literals in $\mathscr{M}$ can be unified with literals of opposite sign in $\mathscr{L}$. If other clauses have literals assigned to $B \rhd D$, they may be partitioned in turn. Alternatively, an SGGS-strategy may simply delete clauses with literals assigned to a partitioned one: such deletions, like those embedded in SGGS-resolution (cf. Definition 26) represent a (partial) *restart* of the search, resembling a restart in CDCL. We use *partitioning inferences* to include splitting inferences and recursive partitioning inferences. A partitioning inference is *trivial* if it replaces a clause by a trivial partition. The following example showcases several SGGS inference rules:

Example 12 Consider the set $S = \{\neg P(f(x)) \vee \neg Q(g(x)) \vee R(x),\ P(x),\ Q(y),\ \neg R(c)\}$ with all-negative I, and a heuristic that selects the rightmost eligible literal whenever there is a choice. The steps of an SGGS-derivation are as follows:

1. Clause $P(x)$ is not satisfied by I, and therefore *non-critical non-conflicting SGGS-extension* adds it to $\Gamma_0 = \varepsilon$ producing $\Gamma_1 = [P(x)]$. $I[\Gamma_1]$ satisfies all ground instances of $P(x)$ and no other positive literal.

2. As clause $Q(y)$ is not satisfied by $I[\Gamma_1]$, another *non-critical non-conflicting SGGS-extension* yields $\Gamma_2 = [P(x)],\ [Q(y)]$, so that $I[\Gamma_2]$ satisfies all ground instances of $P(x)$ and $Q(y)$ and no other positive literal.
3. Since $I[\Gamma_2]$ does not satisfy $\neg P(f(x)) \vee \neg Q(g(x)) \vee R(x)$, a third *non-critical non-conflicting SGGS-extension* generates $\Gamma_3 = [P(x)],\ [Q(y)],\ \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)]$, where $\neg P(f(x))$ is assigned to $P(x)$, $\neg Q(g(x))$ is assigned to $Q(y)$, and $I[\Gamma_3]$ satisfies all ground instances of $P(x)$, $Q(y)$, and $R(x)$, but no other positive literal.
4. As $\neg R(c)$ is in conflict with $I[\Gamma_3]$, *SGGS-extension with I-all-true conflict clause* gives $\Gamma_4 = [P(x)],\ [Q(y)],\ \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$ with $\neg R(c)$ assigned to $R(x)$ and $I[\Gamma_4] = I[\Gamma_3]$. This is the first stage where the disjoint prefix does not coincide with the whole sequence, as $\neg R(c)$ remains outside of $dp(\Gamma_4)$.
5. The conditions for a *left splitting* step are fulfilled, and I-all-true clause $\neg R(c)$ splits clause $\neg P(f(x)) \vee \neg Q(g(x)) \vee R(x)$, yielding $\Gamma_5 = [P(x)],\ [Q(y)],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ \neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)],\ [\neg R(c)]$. Clearly, $\neg R(c)$ is still in conflict with $I[\Gamma_5] = I[\Gamma_4]$.
6. Next *SGGS-move* puts $\neg R(c)$ on the left of its representative in the splitting, namely $\neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)]$, resulting in $\Gamma_6 = [P(x)],\ [Q(y)],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)],\ \neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)]$. The effect is to *flip* the truth value of $\neg R(c)$ from false to true, solving the conflict that it represented, and putting $\neg R(c)$ in the disjoint prefix, as the justification of its sole literal. Now clause $\neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)]$ is in conflict with $I[\Gamma_6]$ and it is not I-all-true.
7. *SGGS-resolution* explains this conflict by resolving the I-false literal $R(c)$ in $\neg P(f(c)) \vee \neg Q(g(c)) \vee [R(c)]$ with the implied literal $\neg R(c)$ that makes it false in $I[\Gamma_6]$. The outcome is $\Gamma_7 = [P(x)],\ [Q(y)],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)],\ \neg P(f(c)) \vee [\neg Q(g(c))]$, where the last clause is I-all-true and in conflict with $I[\Gamma_7] = I[\Gamma_6]$.
8. Thus, *left-splitting* splits the second clause by the last, so as to isolate the intersection between $Q(y)$ and $\neg Q(g(c))$: $\Gamma_8 = [P(x)],\ top(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],\ [Q(g(c))],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)],\ \neg P(f(c)) \vee [\neg Q(g(c))]$ with $I[\Gamma_8] = I[\Gamma_7]$.
9. Now *SGGS-move* places the I-all-true conflict clause $\neg P(f(c)) \vee [\neg Q(g(c))]$ on the left of its representative in the splitting, getting $\Gamma_9 = [P(x)],\ top(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],\ \neg P(f(c)) \vee [\neg Q(g(c))],\ [Q(g(c))],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$. The effect is to *flip* the truth value of $\neg Q(g(c))$ from false to true, solving the conflict represented by $\neg P(f(c)) \vee \neg Q(g(c))$, that enters the disjoint prefix as the justification of its selected literal. Literal $Q(g(c))$ is false in $I[\Gamma_9]$, and because it is the only literal in its clause, we have a conflict clause made of a single I-false literal.
10. *SGGS-resolution* explains this conflict by resolving $Q(g(c))$ with the implied literal $\neg Q(g(c))$ in the disjoint prefix, producing $\Gamma_{10} = [P(x)],\ top(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],\ \neg P(f(c)) \vee [\neg Q(g(c))],\ [\neg P(f(c))],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$, where the resolvent $\neg P(f(c))$ is I-all-true and in conflict with $I[\Gamma_{10}] = I[\Gamma_9]$.

11. To pull out the intersection between $P(x)$ and $\neg P(f(c))$ *left-splitting* applies to split $P(x)$: $\Gamma_{11} = top(x) \neq f \rhd [P(x)],\ y \not\equiv c \rhd [P(f(y))],\ [P(f(c))],\ top(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],\ \neg P(f(c)) \vee [\neg Q(g(c))],\ [\neg P(f(c))],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$ with $I[\Gamma_{11}] = I[\Gamma_{10}]$.
12. This enables *SGGS-move* to move $\neg P(f(c))$ to the left of $P(f(c))$: $\Gamma_{12} = top(x) \neq f \rhd [P(x)],\ y \not\equiv c \rhd [P(f(y))],\ [\neg P(f(c))],\ [P(f(c))],\ top(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],\ \neg P(f(c)) \vee [\neg Q(g(c))],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$. The effect is to *flip* the truth value of $\neg P(f(c))$ from false to true, solving the conflict that $\neg P(f(c))$ represented.
13. The ensuing *SGGS-resolution* of $\neg P(f(c))$ and $P(f(c))$ reveals the inconsistency: $\Gamma_{13} = top(x) \neq f \rhd [P(x)],\ y \not\equiv c \rhd [P(f(y))],\ [\neg P(f(c))],\ \bot,\ top(y) \neq g \rhd [Q(y)],\ z \not\equiv c \rhd [Q(g(z))],\ \neg P(f(c)) \vee [\neg Q(g(c))],\ x \not\equiv c \rhd \neg P(f(x)) \vee \neg Q(g(x)) \vee [R(x)],\ [\neg R(c)]$; and this terminates the derivation.

5 The Lifting Theorem and Other Properties of the SGGS Inference System

In this section we prove two main results about SGGS: first we show that if $\Gamma \neq dp(\Gamma)$, that is, the disjoint prefix is a proper prefix of Γ, an inference rule other than SGGS-extension applies to Γ. Then, we prove a lifting theorem which shows that if $\Gamma = dp(\Gamma)$ and $I[\Gamma] \not\models S$, an SGGS-extension inference rule applies. We begin with two preparatory lemmas. The first one uses the invariant that all literals in a sequence are either I-true or I-false (cf. Condition (1) in Definition 1). In the following we use the notation $\pi_i[\Gamma]$ for the clause of index i in Γ that was introduced after Definition 1.

Lemma 7 *If $\pi_j[\Gamma]$ has a constrained ground instance D' such that $I^p(\Gamma|_{j-1}) \cap D' \neq \emptyset$, splitting by similar literal applies to Γ.*

Proof If $\pi_j[\Gamma] = B \rhd D$ has a constrained ground instance D' such that $I^p(\Gamma|_{j-1}) \cap D' \neq \emptyset$, by Definition 3, there must be a $\pi_i[\Gamma] = A \rhd C[L]$ for some $i < j$, such that some literal M' of D' is in $pcgi(A \rhd L, \Gamma)$. Let M be a literal of D such that M' is instance of M. It follows that L and M have the same sign. Since all literals of all clauses in a sequence are either I-true or I-false, and L and M have the ground instance M' in common, it also follows that either both L and M are I-true or both L and M are I-false. Thus, s-splitting applies to $A \rhd C[L]$ and $B \rhd D\langle M \rangle$. □

Lemma 8 *If the selected literals of $\pi_i[\Gamma]$ and $\pi_j[\Gamma]$ intersect, for some $1 \leq i \neq j \leq |\Gamma|$, an SGGS inference rule other than SGGS-extension applies to Γ.*

Proof Without loss of generality, let $\pi_i[\Gamma] = A \rhd C[L]$ and $\pi_j[\Gamma] = B \rhd D[M]$, where $i < j$, so that C occurs to the left of D, and (i, j) is the smallest pair of indices satisfying the hypothesis, with pairs of indices compared in the lexicographic extension of the ordering on the integers to pairs of integers. Any prefix of $\Gamma|_i$ does not include a pair of intersecting literals, and therefore is a prefix of $dp(\Gamma)$. This means that $A \rhd C[L]$ is in $dp(\Gamma)$, so that $Gr(A \rhd L) = pcgi(A \rhd L, \Gamma)$. By hypothesis, $at(Gr(A \rhd L)) \cap at(Gr(B \rhd M)) \neq \emptyset$. Then, we distinguish two cases:

- $at(Gr(B \rhd M)) \subseteq at(Gr(A \rhd L)) = at(pcgi(A \rhd L, \Gamma))$: if L and M have the same sign, D is disposable and SGGS-deletion applies. Let L and M have opposite sign. If L is I-true and M is I-false, SGGS-resolution applies to C and D. If L is I-false and M is I-true, then M can be assigned, and therefore it is, by Condition (3) in Definition 9. If M were assigned to $\pi_k[\Gamma]$, for some k, $i < k < j$, then (i, j) would not be the smallest pair satisfying the hypotheses, as (i,k) would be smaller. Thus, M is assigned to C, and either SGGS-factoring or left splitting or SGGS-move applies to C and D.
- $at(Gr(B \rhd M)) \not\subseteq at(Gr(A \rhd L))$: if L and M have the same sign and truth value in I, splitting by similar literals applies. If L and M have opposite sign and truth value in I, splitting by dissimilar literals applies. □

Theorem 3 *For all SGGS clause sequences Γ, if $\Gamma \neq dp(\Gamma)$, an SGGS inference rule other than SGGS-extension applies to Γ.*

Proof If $dp(\Gamma)$ is a proper prefix of Γ, it means that there is an i, $1 \leq i \leq |\Gamma|$, such that $\pi_i(\Gamma)$ is not in $dp(\Gamma)$. Let $B \rhd D[M]$ be $\pi_i(\Gamma)$. By Definition 5, $Gr(B \rhd D[M]) \neq pcgi(B \rhd D[M], \Gamma)$, which means there is a $D'[M'] \in Gr(B \rhd D[M])$ such that $D'[M'] \notin pcgi(B \rhd D[M], \Gamma)$. According to Definition 3, there are two cases: either $D' \cap I^p(\Gamma|_{i-1}) \neq \emptyset$ or $D' \in ccgi(B \rhd D[M], \Gamma)$.

- If $D' \cap I^p(\Gamma|_{i-1}) \neq \emptyset$, then by Lemma 7, an s-splitting step applies to Γ.
- If $D'[M'] \in ccgi(B \rhd D[M], \Gamma)$, then $\neg M' \in I^p(\Gamma|_{i-1})$. This means that there exists a j, $j < i$, such that $\pi_j(\Gamma) = A \rhd C[L]$ and $\neg M' \in pcgi(A \rhd L, \Gamma) \subseteq Gr(A \rhd L)$. It follows that M and L intersect, and by Lemma 8, an inference other than SGGS-extension applies to Γ. □

When $\Gamma = dp(\Gamma)$, the lifting theorem says that if a clause $C \in S$ has a ground instance C' such that $I[\Gamma] \not\models C'$, it is possible to extend Γ with a clause $A \rhd E$, such that C' is a constrained ground instance of $A \rhd E$ and E is an instance of C:

Theorem 4 (Lifting Theorem) *Let S be the set of input clauses, I the initial interpretation, Γ an SGGS clause sequence such that $\Gamma = dp(\Gamma)$, and C' a ground instance of some clause $C \in S$ such that $I[\Gamma] \not\models C'$. Then there is a constrained clause $A \rhd E$ such that: (1) C' is a constrained ground instance of $A \rhd E$; (2) E is an instance of C; (3) $A \rhd E$ can be added to Γ by an SGGS-extension inference yielding Γ'; (4) $A \rhd E$ is not disposable in Γ'; and (5) if the SGGS-extension adding $A \rhd E$ is not conflicting, then $pcgi(A \rhd E, \Gamma') \neq \emptyset$.*

Proof $I[\Gamma] \not\models C'$ means that for all $L' \in C'$, either L' is I-true and depends on an I-false selected literal in $\Gamma = dp(\Gamma)$, or L' is I-false and depends on an I-true selected literal in $\Gamma = dp(\Gamma)$, or L' is I-false and $at(L') \notin at(I^p(\Gamma))$. Let μ be the substitution such that $C' = C\mu$ and $\{L_1, \ldots, L_n\}$ be the set of literals of C such that $L_i\mu$, $1 \leq i \leq n$, is I-true. Since $I[\Gamma] \not\models C'$ there must be clauses $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$ in $\Gamma = dp(\Gamma)$, such that for all i, $1 \leq i \leq n$, M_i is I-false and $\neg L_i\mu \in pcgi(B_i \rhd M_i, \Gamma)$. Let $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$ be the leftmost such clauses. For all i, $1 \leq i \leq n$, $\neg L_i$ and $B_i \rhd M_i$ have a (constrained) ground instance in common; since all constrained ground instances of $B_i \rhd M_i$ are ground instances of M_i, $\neg L_i$ and M_i have a ground

instance in common; since $vars(L_i) \cap vars(M_i) = \emptyset$, as they are literals of distinct clauses, it follows that $\neg L_i$ and M_i unify. Also $vars(M_i) \cap vars(M_j) = \emptyset$, for all $1 \leq i \neq j \leq n$, as they are literals of distinct clauses. Thus, there exists a simultaneous most general unifier α such that $L_i\alpha = \neg M_i\alpha$, for all i, $1 \leq i \leq n$. Let β be a most general semantic falsifier for $(C \setminus \{L_1, \ldots, L_n\})\alpha$ such that $C' = C\mu$ is constrained ground instance of $(\bigwedge_{i=1}^{n} B_i\alpha\beta) \rhd C\alpha\beta$. Such a β exists, because C' is false in $I[\Gamma]$ and $\{L_1\mu, \ldots, L_n\mu\}$ are all the I-true literals in C'. Thus, at the very least, β is the substitution such that $C\alpha\beta = C\mu = C'$. Up to here we have proved that the SGGS-extension inference scheme (cf. Definition 12) can generate from main premise C and side premises $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$ a clause $(\bigwedge_{i=1}^{n} B_i\alpha\beta) \rhd C\alpha\beta$ that has $C' = C\mu$ as constrained ground instance. For the rest of the proof, we distinguish three cases corresponding to SGGS-extension with I-all-true conflict clause, SGGS-extension with non-I-all-true conflict clause, and non-conflicting SGGS-extension:

(I) If C' is I-all-true, let $A \rhd E = (\bigwedge_{i=1}^{n} B_i\alpha\beta) \rhd C\alpha\beta$, and τ be the substitution such that $\models \bigwedge_{i=1}^{n} B_i\alpha\beta\tau$ and $C\alpha\beta\tau = C\mu$. Claim (1) is true by construction, as $C' = C\mu$ and $E\tau = C\mu$ imply $C' = E\tau$. Claim (2) is also true by construction, as $E = C\alpha\beta$. For claims (3) and (4), $A \rhd E$ is I-all-true, because it has the I-all-true constrained ground instance C', and all its literals are either I-true or I-false by construction; it is a conflict clause, as all its literals depend on the side premises, which also means that it is not disposable; and can be added by SGGS-extension with I-all-true conflict clause.

(II) If C' has I-false literals, let $\{Q_1, \ldots, Q_k\}$ be the set of literals of C such that $Q_i\mu$, $1 \leq i \leq k$, is I-false. Since $I[\Gamma] \not\models C'$, for all i, $1 \leq i \leq k$, either $Q_i\mu$ depends on an I-true selected literal in $\Gamma = dp(\Gamma)$, or $at(Q_i\mu) \notin at(I^p(\Gamma))$. Assume that for all i, $1 \leq i \leq k$, $Q_i\mu$ depends on an I-true selected literal in $dp(\Gamma)$. Then there are clauses $H_1 \rhd F_1[P_1], \ldots, H_k \rhd F_k[P_k]$ in $dp(\Gamma)$, such that for all i, $1 \leq i \leq k$, P_i is I-true and $\neg Q_i\mu \in pcgi(H_i \rhd P_i, \Gamma)$. The latter condition means that there exist substitutions ρ_i, $1 \leq i \leq k$, such that $\models H_i\rho_i$ and $\neg Q_i\mu = P_i\rho_i$. Since $vars(H_i \rhd P_i) \cap vars(H_j \rhd P_j) = \emptyset$, for all $1 \leq i \neq j \leq k$, this also means that there exists a single substitution ρ such that $\models H_i\rho$ and $\neg Q_i\mu = P_i\rho$, for all i, $1 \leq i \leq k$. We show that there exists an extension substitution for $(\bigwedge_{i=1}^{n} B_i\alpha\beta) \rhd C\alpha\beta$ in Γ. Recall that $C' = C\mu$ is a constrained ground instance of $(\bigwedge_{i=1}^{n} B_i\alpha\beta) \rhd C\alpha\beta$, which means that for all i, $1 \leq i \leq k$, $Q_i\mu$ is a constrained ground instance of $(\bigwedge_{i=1}^{n} B_i\alpha\beta) \rhd Q_i\alpha\beta$. Also, $\{Q_1\alpha\beta, \ldots, Q_k\alpha\beta\}$ are all the I-false literals in $C\alpha\beta$, because $\{Q_1\mu, \ldots, Q_k\mu\}$ are all the I-false literals in $C\mu$. Then, for all i, $1 \leq i \leq k$, $(\bigwedge_{i=1}^{n} B_i\alpha\beta) \rhd \neg Q_i\alpha\beta$ and $H_i \rhd P_i$ have a constrained ground instance in common. Let λ be the most general substitution such that for all i, $1 \leq i \leq k$, $\models H_i\lambda$ and $\neg Q_i\alpha\beta\lambda = P_i\lambda$. Such a substitution exists, because at the very least it is the substitution λ such that $Q_i\alpha\beta\lambda = Q_i\mu$ and $P_i\lambda = P_i\rho$. In practice, λ can be computed as the simultaneous most general unifier of $\neg Q_i\alpha\beta$ and P_i for all i, $1 \leq i \leq k$. Furthermore, λ is an extension substitution, because by construction we have that for all i, $1 \leq i \leq k$, $\neg Gr(\bigwedge_{i=1}^{n} B_i\alpha\beta\lambda \rhd Q_i\alpha\beta\lambda) \subseteq pcgi(H \rhd P, \Gamma)$. Let $A \rhd E = (\bigwedge_{i=1}^{n} B_i\alpha\beta\lambda) \rhd C\alpha\beta\lambda$, and τ be the substitution such that $\models \bigwedge_{i=1}^{n} B_i\alpha\beta\lambda\tau$ and $C\alpha\beta\lambda\tau = C\mu$. Claims (1) and (2) are true by construction, as $C' = C\mu$ and $E\tau = C\mu$ imply $C' = E\tau$; and $E = C\alpha\beta\lambda$. For claims

(3) and (4), $A \rhd E$ is a non-I-all-true conflict clause, because by construction all its I-false literals depend on I-true selected literals in $dp(\Gamma)$; it is not disposable; and can be added by SGGS-extension with non-I-all-true conflict clause.

(III) Otherwise, there is a $Q_i\mu$ in C', for some i, $1 \leq i \leq k$, such that $at(Q_i\mu) \notin at(I^p(\Gamma))$. Let $A \rhd E = (\bigwedge_{i=1}^{n} B_i\alpha\beta) \rhd C\alpha\beta$, and τ be the substitution such that $\models \bigwedge_{i=1}^{n} B_i\alpha\beta\tau$ and $C\alpha\beta\tau = C\mu$. Claims (1) and (2) are true by construction, as in Case (I). For claims (3), (4), and (5), first we observe that since $Q_i\mu$ is a constrained ground instance of $A \rhd Q_i\alpha\beta$, $at(Q_i\mu) \notin at(I^p(\Gamma))$ implies that $at(Gr(A \rhd Q_i\alpha\beta)) \not\subseteq at(pcgi(H \rhd P, \Gamma))$ for all selected literals $H \rhd P$ in Γ, so that $A \rhd E$ has at least one literal, namely $A \rhd Q_i\alpha\beta$, that satisfies the condition to be selected in SGGS-extension with non-conflicting clause. Then, let L be $Q_i\alpha\beta$. The condition $at(Q_i\mu) \notin at(I^p(\Gamma))$ also implies that $A \rhd L$ has at least a proper constrained ground instance, namely $Q_i\mu$ itself. Thus, $A \rhd E[L]$ has at least a proper constrained ground instance, namely C', it is not disposable, and can be added by SGGS-extension with non-conflicting clause, which will be critical or not depending on the existence of a proper prefix of Γ enabling a critical step. □

Since α, β, and γ are most general substitutions, $A \rhd E$ is the most general clause such that $C' \in Gr(A \rhd E)$, E is instance of C, every literal in E is either I-true or I-false, and if all I-false literals in E intersect I-true selected literals then they depend on them. To exemplify the lifting theorem we continue Example 6:

Example 13 With $S = \{P(a), \neg P(x) \vee Q(f(y)), \neg P(x) \vee \neg Q(z)\}$, and I all-negative, we were left with $\Gamma_2 = [P(a)], \neg P(a) \vee [Q(f(y))]$. Since $I[\Gamma_2] \models P(a)$ and $I[\Gamma_2] \models Q(f(t))$ for all ground terms t, instance $C' = \neg P(a) \vee \neg Q(f(f(a)))$ of $C = \neg P(x) \vee \neg Q(z)$, with $\mu = \{x \leftarrow a, z \leftarrow f(f(a))\}$, is false in $I[\Gamma_2]$. An SGGS-extension with I-all-true conflict clause extends Γ_2 with $E = \neg P(a) \vee [\neg Q(f(w))]$, obtained by applying $\alpha = \{x \leftarrow a, z \leftarrow f(w)\}$ to unify the I-true literals of C with the two I-false selected literals in Γ_2. Clause C' is a ground instance of E with $\tau = \{w \leftarrow f(a)\}$. Thus, we have $\Gamma_3 = [P(a)], \neg P(a) \vee [Q(f(y))], \neg P(a) \vee [\neg Q(f(w))]$, where $\neg P(a)$ is assigned to $P(a)$, $\neg Q(f(w))$ is assigned to $Q(f(y))$, and $\neg Q(f(w))$ is selected so that the selected literal in E is assigned rightmost. SGGS-move yields $\Gamma_4 = [P(a)], \neg P(a) \vee [\neg Q(f(w))], \neg P(a) \vee [Q(f(y))]$, and SGGS-resolution generates $\Gamma_5 = [P(a)], \neg P(a) \vee [\neg Q(f(w))], [\neg P(a)]$, where the last clause is I-all-true and in conflict with $I[\Gamma_5]$. SGGS-move gets $\Gamma_6 = [\neg P(a)], [P(a)], \neg P(a) \vee [\neg Q(f(w))]$ and SGGS-resolution closes the refutation with $\Gamma_7 = [\neg P(a)], \bot, \neg P(a) \vee [\neg Q(f(w))]$.

Theorems 3 and 4 together ensure that the system can make progress. The following characterization of an SGGS-derivation ensures that it does:

Definition 32 (Sensible SGGS-derivation) Given set S of clauses and initial interpretation I, an SGGS-derivation from S is *sensible*, if (i) no partitioning inference is trivial, (ii) SGGS-deletion is applied eagerly, (iii) SGGS-extension is applied whenever $I[\Gamma] \not\models S$, $\bot \notin \Gamma$, and $\Gamma = dp(\Gamma)$, and (iv) an inference other than SGGS-extension is applied whenever $I[\Gamma] \not\models S$, $\bot \notin \Gamma$, and $\Gamma \neq dp(\Gamma)$.

Indeed, there is no need to continue after success. An SGGS-strategy is *sensible* if all its derivations are. From now on we consider sensible derivations and strategies.

6 Conflict Explanation and Conflict Solving in SGGS

In this section we complete the illustration of how SGGS handles conflicts and lifts CDCL to first order. In Section 4 we saw how SGGS-move corresponds to backjumping, and admitting a former conflict clause to the disjoint prefix is the counterpart of learning it. Here we add that SGGS-resolution *explains* a first-order conflict like propositional resolution does for a propositional one. We define *explanation inferences* and *solving inferences* for a conflict clause, and we characterize a conflicting SGGS-extension as *bundled*, if it is followed by explanation and solving inferences for its conflict clause. The name comes from the intuition that conflicting SGGS-extension, possibly explanation inferences, and solving inferences are applied in a bundle. In the second part of the section we introduce *bundled splitting inferences*, and we expand the notions of sensible SGGS-derivation and SGGS-strategy into those of *bundled* SGGS-derivation and SGGS-strategy.

A conflicting SGGS-extension appends a conflict clause $A \rhd E[L]$ to Γ. If $E[L]$ is I-all-true, the *solving inferences* use E to *left-split* the clause D that L is assigned to, and then *move* E to the left of its representative in $split(D,E)$, so that the truth value in $I[\Gamma]$ of the constrained ground instances of L is *flipped*, and the conflict is solved. Preliminarly, the system checks whether SGGS-factoring applies to E, and if yes, it is the factor that left-splits D and moves to the left to solve the conflict:

Definition 33 (Solving inferences) The *solving inferences* for I-all-true conflict clause $A \rhd E[L]$ in $\Gamma\ A \rhd E[L]$, where L is assigned to $B \rhd D[M]$ in $dp(\Gamma)$, consist of

1. SGGS-factoring applied to D and E until no longer applicable, and resulting in factor E_f, which is E itself, if no SGGS-factoring applies;
2. Left splitting applied to split D by E_f, if applicable; and
3. SGGS-move applied to move E_f to the left of its representative in $split(D,E_f)$, if left splitting applied, or to the left of D otherwise.

Left splitting is not needed if $at(L)$ and $at(M)$ are identical as in the instances of SGGS-move in Example 10, while every SGGS-move in Example 12 is preceded by left splitting. If $E[L]$ is not I-all-true, *explanation inferences* resolve away by SGGS-resolution *all* I-false literals in E to get either $\bot$ or an I-all-true clause:

Definition 34 (Explanation inferences) The *explanation inferences* for a non-I-all-true conflict clause $A \rhd E[L]$ in $\Gamma\ A \rhd E[L]$, where $H_1 \rhd F_1[P_1]$ is the I-all-true clause in $dp(\Gamma)$ such that $A \rhd L$ depends on $H_1 \rhd P_1$, consist of a series of SGGS-resolution steps
$res(E[L], F_1[P_1], E_1[L_1])$
$res(E_1[L_1], F_2[P_2], E_2[L_2])$
...
$res(E_{n-1}[L_{n-1}], F_n[P_n], E_n[L_n])$
where, for all i, $1 \leq i < n$, no E_i is I-all-true, $F_{i+1}[P_{i+1}]$ is the I-all-true clause in $dp(\Gamma)$ such that L_i depends on P_{i+1}, and E_n is either $\bot$ or an I-all-true clause.

This series of SGGS-resolution steps corresponds to the propositional resolution steps that either *explain* a conflict or find a refutation in CDCL (e.g., [49,51,48,46]).

In CDCL, a propositional clause C is a conflict clause because for all its literals L, $\neg L$ appears in the trail; $\neg L$ may be either decided or implied, justified by another clause D, whose literals other than $\neg L$ are falsified by the trail. Explanation resolves upon L in C and $\neg L$ in D, generating a new conflict clause. For SGGS, consider a conflict clause E which is not I-all-true: by Definition 12 all I-true literals in E are assigned, and by Definition 19 each I-false literal in E depends on an I-true selected literal in $dp(\Gamma)$. An I-true literal is selected in an I-all-true clause, and an I-all-true clause in $dp(\Gamma)$ is the justification of its selected literal: all its literals are assigned except the selected literal which is the implied literal. Thus, SGGS *explains* the conflict by resolving conflict clauses with justifications like CDCL: every SGGS-resolution step in the series of explanation inferences resolves a conflict clause with a justification, resolving upon an I-false literal in the conflict clause and the implied literal in the justification. Every resolvent is a conflict clause that replaces the previous one at the last index of the sequence. If the empty clause arises, a refutation is found. Otherwise, the resulting I-all-true conflict clause is subject to the solving inferences:

Definition 35 (Bundled SGGS-extension) A conflicting SGGS-extension with I-all-true conflict clause $A \rhd E[L]$ is *bundled*, if it is followed by the solving inferences for $A \rhd E[L]$. A conflicting SGGS-extension with non-I-all-true conflict clause $A \rhd E[L]$ is *bundled*, if it is followed by the explanation inferences for $A \rhd E[L]$, and the solving inferences for $A' \rhd E'[L']$, if the explanation inferences for $A \rhd E[L]$ yield non-empty I-all-true conflict clause $A' \rhd E'[L']$.

In CDCL, how many resolutions to do is a matter of heuristic. The last resolvent is learned and used to modify the trail by backjumping: one of its literals switches from false to true, and enters the trail as implied literal justified by the learned clause. In the *first unique implication point* heuristic, the learned clause is called *asserting clause*. In SGGS, explanation by resolution resolves away *all* I-false literals in the conflict clause, because the application of the extension substitution (cf. Definition 19) ensures that they *all* depend on implied literals. Possibly after SGGS-factoring and left-splitting, the resulting I-all-true conflict clause moves left and enters the disjoint prefix: this I-all-true clause corresponds to the asserting clause. In the following we use *bundled SGGS-extension* (with extension clause E) for the macro inference formed by conflicting SGGS-extension (with extension clause E), possibly explanation inferences, and solving inferences. Deletion of disposable clauses applies at the end rather than during a bundled SGGS-extension.

Also splitting inferences need to be followed by appropriate steps:

Definition 36 (Bundled splitting) A splitting inference is *bundled*, if it is followed by either all applicable recursive partitioning steps or the deletion of all clauses with literals assigned to the splitted clause.

As observed in Section 4 (cf. Definition 31), these deletions mirror a (partial) *restart* in CDCL. Sensibility of a derivation ensures that SGGS makes progress; bundledness specifies this further by ensuring that conflicts are solved eagerly:

Definition 37 (Bundled SGGS-derivation) An SGGS-derivation is *bundled*, if it is sensible, all conflicting SGGS-extension inferences are bundled, and all splitting inferences are bundled.

An SGGS-strategy is *bundled* if all its derivations are. To exemplify bundled SGGS-extension, we continue Example 10:

Example 14 Suppose that the first SGGS-extension with non-I-all-true conflict clause is applied, yielding $\Gamma_9 = [P(a)],\ [\neg R(y,x)],\ [P(f(x))],\ [\neg Q(x)],\ [\neg P(a)] \vee R(z,f(z)) \vee Q(a)$. In a bundled SGGS-derivation, this conflict is handled prior to considering applying any other SGGS-extension. SGGS-resolution explains the conflict resolving the only I-false literal in the conflict clause, namely $\neg P(a)$, with the implied literal $P(a)$ in $dp(\Gamma)$, yielding $\Gamma_{10} = [P(a)],\ [\neg R(y,x)],\ [P(f(x))],\ [\neg Q(x)],\ [R(z,f(z))] \vee Q(a)$. The I-all-true resolvent $[R(z,f(z))] \vee Q(a)$ left-splits $\neg R(y,x)$, generating $\Gamma_{11} = [P(a)],\ [\neg R(y,f(y))],\ x \not\equiv f(y) \rhd [\neg R(y,x)],\ [P(f(x))],\ [\neg Q(x)],\ [R(z,f(z))] \vee Q(a)$, and then moves left, flipping the truth value of all ground instances of $R(z,f(z))$ in the induced interpretation: $\Gamma_{12} = [P(a)],\ [R(z,f(z))] \vee Q(a),\ [\neg R(y,f(y))],\ x \not\equiv f(y) \rhd [\neg R(y,x)],\ [P(f(x))],\ [\neg Q(x)]$. This completes the bundled inference. Note that at this stage the second SGGS-extension with non-I-all-true conflict clause of Example 10, namely SGGS-extension with $\neg P(f(x)) \vee R(z,f(z)) \vee Q(f(x))$ is no longer applicable, because, as an effect of the SGGS-move step, $I[\Gamma_{12}]$ satisfies this clause. Next, SGGS-resolution amends the sequence with respect to the move, by resolving $[R(z,f(z))] \vee Q(a)$ with $[\neg R(y,f(y))]$ to give $\Gamma_{13} = [P(a)],\ [R(z,f(z))] \vee Q(a),\ [Q(a)],\ x \not\equiv f(y) \rhd [\neg R(y,x)],\ [P(f(x))],\ [\neg Q(x)]$. The resolvent $Q(a)$ is also I-all-true and in $dp(\Gamma)$, and therefore SGGS-resolves with $[\neg Q(x)]$ to close the refutation: $\Gamma_{14} = [P(a)],\ [R(z,f(z))] \vee Q(a),\ [Q(a)],\ x \not\equiv f(y) \rhd [\neg R(y,x)],\ [P(f(x))],\ \bot$. Thus, a bundled SGGS-strategy finds a refutation without applying the second SGGS-extension with non-I-all-true conflict clause.

By focusing on one conflict at a time, bundledness may help to focus on a proof and apply SGGS-extension sparingly. During explanation and solving inferences up to SGGS-move, the presence of a conflict clause at the rightmost end of the sequence means that $dp(\Gamma) \neq \Gamma$, so that by sensibility SGGS-extension does not apply, and by bundledness the explanation and solving inferences apply. In both CDCL and SGGS conflict solving has priority over extension of the candidate model by decision or SGGS-extension and literal selection, respectively. The following lemma shows how bundled SGGS-extension effectively changes the SGGS clause sequence:

Lemma 9 *Let Γ' be derived from Γ by a bundled SGGS-extension with extension clause E in a bundled SGGS-derivation. Then either Γ' contains $\bot$, or there exists an index i, $i > 0$, such that $\Gamma = \Gamma_1 D \Gamma_2$, where $\pi_i(\Gamma) = D[M]$, $\Gamma' = \Gamma_1 C \Gamma_3$, where $\pi_i(\Gamma') = C$, and $D[M]$ and C are not equivalent.*

Proof The initial conflicting SGGS-extension transforms Γ into ΓE. If E is not I-all-true, let E' be the clause resulting from the explanation inferences for E. By Definition 34, E' is either $\bot$ or I-all-true. If E is I-all-true, let E' be E itself. Either way, E' is either $\bot$ or I-all-true. If E' is $\bot$, Γ' contains $\bot$ and the claim holds. Otherwise, all literals in $tlits(E')$ are assigned, because all those in $tlits(E)$, if any, are assigned by Definition 12, and SGGS-resolution lets resolvents inherit assignments (cf. Definition 26). Let $\mathscr{I}'$ be the set of indices which the literals in $tlits(E')$ are assigned to, and let i be its maximum. Since all literals in $tlits(E')$ are assigned (E' is an I-all-true conflict clause), the selected literal of E' is assigned, and since the assigned

literal is assigned rightmost (cf. Definition 9), i is the index which the selected literal of E' is assigned to. Let $D[M]$ be $\pi_i(\Gamma)$. Note that the explanation inferences, if any, transform ΓE into $\Gamma E'$, so that Γ is unaffected. Let E_f be the I-all-true conflict clause resulting from the SGGS-factoring steps in the solving inferences for E'. If no SGGS-factoring step applies, let E_f be E' itself. All literals in $tlits(E_f)$ are assigned, because all those in $tlits(E')$ are, and SGGS-factoring lets factors inherit assignments (cf. Definition 27). In particular, the selected literal of E_f is still assigned to i, and i is still the largest index which some literal in $tlits(E_f)$ is assigned to. The left splitting in the solving inferences applies to $D[M]$ and E_f, replacing $D[M]$ by $split(D[M], E_f) = D_1 \dots D_k$, so that D_1 has index i. Let D_j be the representative of E_f in $split(D[M], E_f)$: the SGGS-move in the solving inferences moves E_f to the left of D_j. There are two cases:

- If D_j is D_1, Γ' has E_f at index i, so that the clause C of the claim is E_f. This case covers also the situation where left splitting does not apply and E_f moves to the left of $D[M]$. Then C and $D[M]$ are not equivalent, because it cannot be $Gr(C) = Gr(D[M])$, since C is I-all-true, which means its constrained ground instances are I-all-true, whereas M is I-false, which means the constrained ground instances of $D[M]$ are not I-all-true.
- If D_j is not D_1, Γ' has D_1 at index i, so that C is D_1. Then C and $D[M]$ are not equivalent, because $Gr(C) \subset Gr(D[M])$, since the splitting step that generated $D_1 \dots D_k$ is not trivial, as the derivation is bundled. □

From now on we consider bundled derivations and strategies.

7 Constraints: Standardization and Splitting

This section presents constraint manipulation rules to reduce constraints to standard form and split a clause by another one. These rules are *sound*, meaning that premise and conclusion represent the same set of constraint ground instances. A conclusion of the form $A_1 \rhd C_1, \dots, A_n \rhd C_n$ is understood as disjunction, so that the set of represented constraint ground instances is $\bigcup_{i=1}^{n} Gr(A_i \rhd C_i)$. If a constraint is found unsatisfiable, the result is a clause of the form $false \rhd C$, that has no constraint ground instances ($Gr(false \rhd C) = \emptyset$) hence is trivially true, so that it can be read as $\top$.

7.1 Standardization

The rules for reduction to standard form comprise rules for identity and rules for top symbol. The *rules for identity* eliminate or decompose all identity constraints, except those in the form $x \not\equiv y$:

Definition 38 (Rules for identity) The *rules for identity* are:

- The *ElimId1 rule* eliminates a constraint between variable and term:
  1. If $x \notin vars(s)$, then:

$$\frac{(A \wedge x \equiv s) \rhd C}{(A \rhd C)\{x \leftarrow s\}}$$

2. If $x \in vars(s)$ and s is not a variable, then:

$$\frac{(A \wedge x \equiv s) \rhd C}{false \rhd C} \quad \frac{(A \wedge x \not\equiv s) \rhd C}{(A \rhd C)}$$

– The *ElimId2 rule* detects a conflict: if $f \neq g$, $m \geq 0$, $n \geq 0$, then:

$$\frac{(A \wedge f(s_1, \ldots, s_n) \equiv g(t_1, \ldots, t_m)) \rhd C}{false \rhd C}$$

– The *ElimId3 rule* eliminates a satisfied constraint: if $f \neq g$, $m \geq 0$, $n \geq 0$, then:

$$\frac{(A \wedge f(s_1, \ldots, s_n) \not\equiv g(t_1, \ldots, t_m)) \rhd C}{A \rhd C}$$

– The *ElimId4 rule* decomposes an identity: if $n \geq 0$, then:

$$\frac{(A \wedge f(s_1, \ldots, s_n) \equiv f(t_1, \ldots, t_n)) \rhd C}{(A \wedge s_1 \equiv t_1 \wedge \ldots \wedge s_n \equiv t_n) \rhd C}$$

– The *ElimId5 rule* decomposes a negated identity: if $n \geq 0$, then:

$$\frac{(A \wedge f(s_1, \ldots, s_n) \not\equiv f(t_1, \ldots, t_n)) \rhd C}{(A \wedge (s_1 \not\equiv t_1 \vee \ldots \vee s_n \not\equiv t_n)) \rhd C}$$

– The *ElimId6 rule* eliminates a negated identity between variable and non-variable term:

$$\frac{(A \wedge f(s_1, \ldots, s_n) \not\equiv x) \rhd C}{A \wedge top(x) \neq f \rhd C,\ ((A \wedge f(s_1, \ldots, s_n) \not\equiv f(y_1, \ldots, y_n)) \rhd C)\{x \leftarrow f(y_1, \ldots, y_n)\}}$$

where $n \geq 0$, and the y_i's, $1 \leq i \leq n$, are new variables;

– The *ElimId7 rule* detects a conflict: if s is a variable or constant, then:

$$\frac{(A \wedge s \not\equiv s) \rhd C}{false \rhd C}$$

If Case (1) of ElimId1 applies to $x \equiv y$, one of the two possible replacements (e.g., $x \leftarrow y$) is chosen. The ElimId5 rule calls for restoration of disjunctive normal form:

Definition 39 (DNF rules) The *disjunctive normal form (DNF) rules* are:

– The *Equiv rule* replaces a constraint A by its disjunctive normal form, denoted $dnf(A)$:

$$\frac{A \rhd C}{dnf(A) \rhd C}$$

– And the *Div rule* subdivides disjunction:

$$\frac{(A \vee B) \rhd C}{A \rhd C,\ B \rhd C}$$

If every application of ElimId5 is followed by an application of the DNF rules, first Equiv and then Div, $(A \wedge f(s_1,\ldots,s_n) \not\equiv f(t_1,\ldots,t_n)) \rhd C$ is replaced by $A \wedge s_1 \not\equiv t_1 \rhd C, \ldots, A \wedge s_n \not\equiv t_n \rhd C$. The *rules for top symbol* eliminate all top symbol constraints, except those in the form $top(x) \neq f$:

Definition 40 (Rules for top symbol) The *rules for top symbol* are

- The *ElimTop1 rule* detects a conflict in a positive constraint: if $f \neq g$, $n \geq 0$, then:

$$\frac{A \wedge top(f(s_1,\ldots,s_n)) = g \rhd C}{false \rhd C}$$

- The *ElimTop2 rule* eliminates a satisfied positive constraint: if $n \geq 0$, then:

$$\frac{A \wedge top(f(s_1,\ldots,s_n)) = f \rhd C}{A \rhd C}$$

- The *ElimTop3 rule* eliminates a satisfied negative constraint: if $f \neq g$, $n \geq 0$, then:

$$\frac{A \wedge top(f(s_1,\ldots,s_n)) \neq g \rhd C}{A \rhd C}$$

- The *ElimTop4 rule* detects a conflict in a negated constraint: if $n \geq 0$, then:

$$\frac{A \wedge top(f(s_1,\ldots,s_n)) \neq f \rhd C}{false \rhd C}$$

- The *ElimTop5 rule* eliminates a positive constraint: if $n \geq 0$, then:

$$\frac{A \wedge top(x) = f \rhd C}{(A \rhd C)\{x \leftarrow f(x_1,\ldots,x_n)\}}$$

  where the x_i's, $1 \leq i \leq n$, are new variables.

The combined effect of rules for identity and rules for top symbol is to standardize any constraint:

Definition 41 (Standardization rules) The *standardization rules* comprise the rules for identity and the rules for top symbol, with the provision that every application of ElimId5 is followed by an application of the DNF rules, first Equiv and then Div.

The next example shows that the application of the identity rules may not terminate, if literals are allowed to grow in size:

Example 15 Consider a clause $(x \not\equiv f(y) \wedge y \not\equiv f(x) \rhd P(x,y))$: an application of ElimId6 yields the two clauses $(top(x) \neq f \wedge y \not\equiv f(x)) \rhd P(x,y)$ and $(f(z) \not\equiv f(y) \wedge y \not\equiv f(f(z)) \rhd P(f(z),y))$. Using ElimId5, the latter clause becomes $(z \not\equiv y \wedge y \not\equiv f(f(z)) \rhd P(f(z),y))$, which by another application of ElimId6, yields the two clauses $(z \not\equiv y \wedge top(y) \neq f) \rhd P(f(z),y))$ and $(z \not\equiv f(w) \wedge f(w) \not\equiv f(f(z)) \rhd P(f(z),f(w)))$. Using ElimId5 again, the latter clause becomes $(z \not\equiv f(w) \wedge w \not\equiv f(z) \rhd P(f(z),f(w)))$, whose constraint is a variant of the original one, so that the entire process can be repeated, while the size of the constrained literal has increased.

On the other hand, the following lemma, invoked in the proof of Theorem 6 in Section 8, shows that there is no non-termination without growth in size:

Lemma 10 *Let $\mathscr{C}$ be a set of constrained literals under a fixed upper bound in size: $\mathscr{C} = \{A \rhd L : |L| \leq n,\ n > 0\}$. If the standardization of $A \rhd L \in \mathscr{C}$ does not terminate, it must produce a constrained literal $B \rhd M \notin \mathscr{C}$.*

Proof Let the size of a constraint A be given by the number of occurrences of constant and function symbols in A. By inspection, the application of standardization rules to $A \rhd L$ is guaranteed to terminate, because it reduces either the size of A or the size of L, except for those rules that replace variables by terms, namely Case (1) of ElimId1, ElimId6, and ElimTop5. However, these rules cannot be applied an unbounded number of times without generating eventually a literal that is not in $\mathscr{C}$ because its size exceeds n. □

We consider next the computation of clause splitting.

7.2 Splitting and Difference

In order to compute $split(C,D)$, for $A \rhd C\langle L\rangle$ and $B \rhd D[M]$ as in Definition 14, we need to compute the representative of D in $split(C,D)$, and the other clauses in the partition of C: we call the latter *difference*, denoted $C - D$.

Example 16 For $C = true \rhd P(x,y)$ and $D = true \rhd P(f(w),g(z))$ as in Example 8, the difference $C - D$ is $\{top(x) \neq f \rhd P(x,y),\ top(y) \neq g \rhd P(f(x),y)\}$.

The representative of D in $split(C,D)$ is $A\sigma \wedge B\sigma \rhd C\langle L\rangle\sigma$, where σ is the most general unifier of $at(L)$ and $at(M)$ and $(A \wedge B)\sigma$ is satisfiable. Thus, $split(C,D) = \{A\sigma \wedge B\sigma \rhd C[L]\sigma\} \cup (C - D)$. If $at(L)$ and $at(M)$ do not unify, or $(A \wedge B)\sigma$ is unsatisfiable, there is no intersection between $A \rhd L$ and $B \rhd M$ and no point in splitting $A \rhd C\langle L\rangle$ by $B \rhd D[M]$. For brevity, let $H \rhd F$ denote $A\sigma \wedge B\sigma \rhd C\langle L\rangle\sigma$, so that $split(C,D) = \{H \rhd F\} \cup (C - D)$. By definition of splitting, $split(C,D) = split(C,F)$ and $split(C,F) = \{H \rhd F\} \cup (C - F)$, so that $(C - D) = (C - F)$. In other words, in order to compute the difference between C and D it suffices to compute the difference between C and F. Thus, we only need to compute clause differences of the form $C - C\sigma$, where the second clause is an instance of the first. In the following, we call clauses *similar*, if they are made identical by a substitution that replaces variables by variables, but may replace distinct variables by the same:

Definition 42 (Rules for clause difference) Given clauses $A \rhd C$ and $B \rhd D$, such that $D = C\sigma$ for some substitution σ, the *rules for clause difference* are:

- If for some $x \in vars(C)$ and new variables x_i, $1 \leq i \leq n$, $x \leftarrow f(x_1,\ldots,x_n) \in \sigma$, the *DiffSim rule* applies $\{x \leftarrow f(x_1,\ldots,x_n)\}$ to make C similar to D and on the other hand adds $top(x) \neq f$ to make them different:

$$\frac{(A \rhd C) - (B \rhd D)}{(A \rhd C)\{x \leftarrow f(x_1,\ldots,x_n)\} - (B \rhd D),\ A \wedge (top(x) \neq f) \rhd C}$$

– If C and D are similar, and for distinct variables $x, y \in vars(C)$, $x \leftarrow y \in \sigma$, the *DiffVar rule* applies $\{x \leftarrow y\}$ to make C a variant of D and on the other hand adds $x \not\equiv y$ to make them different:

$$\frac{(A \rhd C) - (B \rhd D)}{(A \rhd C)\{x \leftarrow y\} - (B \rhd D),\ (x \not\equiv y \wedge A) \rhd C}$$

– If C and D are variants but not identical, the *DiffId rule* makes them identical:

$$\frac{(A \rhd C) - (B \rhd D)}{(A \rhd C)\sigma - (B \rhd D)}$$

– And the *DiffElim rule* replaces difference by negation:

$$\frac{(A \rhd C) - (B \rhd C)}{(A \wedge \neg B) \rhd C}$$

Since B is a conjunction of constraints, $\neg B$ is a disjunction of their negations, and therefore every application of DiffElim is followed by restoration of disjunctive normal form by the DNF rules.

Example 17 Assume that we split $P(x,y)$ by $P(a,y)$, where $P(a,y)$ is an instance of $P(x,y)$ with $\sigma = \{x \leftarrow a\}$. $P(a,y)$ is its own representative in the splitting, and the difference $P(x,y) - P(a,y)$ is computed by an application of DiffSim that produces $\{P(a,y) - P(a,y),\ top(x) \neq a \rhd P(x,y)\}$, hence $\{top(x) \neq a \rhd P(x,y)\}$. Thus, the splitting is $\{P(a,y),\ top(x) \neq a \rhd P(x,y)\}$. Symmetrically, if we split $P(x,y)$ by $top(x) \neq a \rhd P(x,y)$, which is its own representative in the splitting, the difference $P(x,y) - top(x) \neq a \rhd P(x,y)$ is computed by an application of DiffElim that yields $top(x) = a \rhd P(x,y)$, that is, $P(a,y)$.

Example 18 If we split $P(x,y)$ by $P(a,b)$, where $P(a,b)$ is an instance of $P(x,y)$ with $\sigma = \{x \leftarrow a, y \leftarrow b\}$, $P(a,b)$ is its own representative in the splitting, and the difference $P(x,y) - P(a,b)$ is computed by two applications of DiffSim, that produce first $\{P(a,y) - P(a,b),\ top(x) \neq a \rhd P(x,y)\}$ and then $\{P(a,b) - P(a,b),\ top(y) \neq b \rhd P(a,y),\ top(x) \neq a \rhd P(x,y)\}$, hence $\{top(y) \neq b \rhd P(a,y),\ top(x) \neq a \rhd P(x,y)\}$. Thus, the splitting is $\{P(a,b),\ top(x) \neq a \rhd P(x,y),\ top(y) \neq b \rhd P(a,y)\}$. If we split $P(x,y)$ by $top(y) \neq b \rhd P(a,y)$, which is its own representative in the splitting, the difference $P(x,y) - top(y) \neq b \rhd P(a,y)$ is computed by an application of DiffSim that generates $\{P(a,y) - top(y) \neq b \rhd P(a,y),\ top(x) \neq a \rhd P(x,y)\}$, and an application of DiffElim that yields $\{top(y) = b \rhd P(a,y),\ top(x) \neq a \rhd P(x,y)\}$, that is, $\{P(a,b),\ top(x) \neq a \rhd P(x,y)\}$.

The following theorem shows that the computation of clause difference halts, so that pathological cases such as Example 15 cannot arise in splitting inferences:

Theorem 5 *Given $A \rhd C$ and $B \rhd D$, such that $D = C\sigma$, and A and B are in standard form, any application of the clause difference rules to $C - D$, where (1) any application of DiffElim or ElimId5 is followed by conversion to DNF, and (2) all constraints are restored to standard form after every application of a clause difference rule, is guaranteed to terminate.*

Proof First we show that the rules for clause difference do not cause non-termination. DiffId and DiffElim can be applied only once. DiffVar can be applied only a finite number of times, because each application decreases the number of variables in C. Each DiffSim step applies to C a substitution $\{x \leftarrow f(x_1, \ldots x_n)\}$ from σ: since σ contains finitely many such pairs, DiffSim can be applied only a finite number of times. Then we prove that standardization between an application of a clause difference rule and the next is guaranteed to terminate:

1. DiffId only renames variables, which does not enable any other rule.
2. DiffVar adds an $x \not\equiv y$, which is in standard form, and applies a substitution $\{x \leftarrow y\}$, whose only effect may be to replace an $x \not\equiv y$ by an $x \not\equiv x$, eliminated by ElimId7.
3. DiffSim adds a $top(x) \neq f$, which is in standard form, and applies a substitution $\{x \leftarrow f(x_1, \ldots, x_n)\}$, which may have two effects. One is to replace the occurrence of x in a constraint $top(x) \neq g$ by $f(x_1, \ldots, x_n)$. This enables either ElimTop3 or ElimTop4, which terminate. The other is to transform an $x \not\equiv y$ into an $f(x_1, \ldots, x_n) \not\equiv y$, or $y \not\equiv f(x_1, \ldots, x_n)$, since $\equiv$ is symmetric, which enables ElimId6. This rule adds a $top(x) \neq f$, which is in standard form, and applies another substitution of the same form, so that eventually a subset of the variables may be replaced by terms $f(x_1, \ldots, x_n)$ where the x_i's are new. This can only be done a finite number of times, because the new variables will never be replaced in this way. If two such substitutions are applied to a $z \not\equiv w$, an $f(x_1, \ldots, x_n) \not\equiv f(y_1, \ldots, y_n)$ may arise. ElimId5 applies to such a constraint, followed by conversion to DNF. The result is a disjunction of constrained clauses, each containing in its constraint an $x_i \not\equiv y_i$, for some i, which is in standard form.
4. DiffElim yields $(A \wedge \neg B) \rhd C$, followed by conversion to DNF. The effect may be to add $x \equiv y$ (negation of $x \not\equiv y$ in B) or $top(x) = f$ (negation of $top(x) \neq f$ in B). In the first case, ElimId1 applies $\{x \leftarrow y\}$, covered in Case (2) of this proof. In the second case, ElimTop5 applies $\{x \leftarrow f(x_1, \ldots, x_n)\}$, covered in Case (3) of this proof. □

8 Completeness and Goal-Sensitivity

In this section we prove that SGGS is *model complete* and *refutationally complete*, regardless of the choice of initial interpretation. These results involve a *convergence ordering*, denoted $>^c$, on SGGS clause sequences. Its key property is that it is *well-founded* on sequences of bounded length; as a special case, it is well-founded on sequences of fixed length, so that there is no infinite descending chain of sequences all of the same length. It follows that every fixed length prefix of the sequences in an SGGS-derivation that is a non-ascending chain (i.e., $\Gamma_0 \geq^c \Gamma_1 \geq^c \ldots \Gamma_j \geq^c \ldots$) eventually reaches a limit, meaning that it does not change anymore. Such a derivation admits *limiting sequence*. The convergence ordering is also used to define *fairness* of an SGGS-derivation, as an extension of bundledness. A central result is the *descending chain theorem*, which proves that a fair SGGS-derivation forms a descending chain, so that it has limiting sequence. The *model completeness theorem* shows that if

S is satisfiable, the interpretation induced by the limiting sequence of any fair SGGS-derivation from S is a model. The *refutational completeness theorem* shows that if S is unsatisfiable, *any fair SGGS-derivation from S is a refutation*: if the empty clause is not generated, the derivation is infinite and without limiting sequence, because there is always another SGGS inference that modifies the sequence, contradicting the existence of a limiting sequence. We close the section by showing that SGGS is *goal-sensitive*, if the initial interpretation is goal-sensitive.

8.1 Convergence Ordering and Fairness

We construct an ordering on SGGS clause sequences. Since the purpose is to compare sequences in a derivation, all sequences to be compared have the same initial interpretation I. Let Λ be a new 0-ary predicate symbol that will be used as a sort of sentinel value in the definition of the ordering. We begin by associating a measure to every clause in a sequence:

Definition 43 (Clause measure) Given $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$, for all i, $1 \leq i \leq n$, the *measure of clause* $A_i \rhd C_i[L_i]$ in Γ, denoted $V_i(\Gamma)$, is defined as follows:

$$V_i(\Gamma) = \begin{cases} 0 \text{ if } A_i \rhd C_i[L_i] \text{ is } \bot, \\ 1 \text{ if } pcgi(A_i \rhd C_i[L_i], \Gamma) \neq \emptyset \wedge L_i \in tlits(C_i), \\ 2 \text{ if } pcgi(A_i \rhd C_i[L_i], \Gamma) \neq \emptyset \wedge L_i \in flits(C_i), \\ 3 \text{ if } pcgi(A_i \rhd C_i[L_i], \Gamma) = \emptyset \wedge ccgi(A_i \rhd C_i[L_i], \Gamma) \neq \emptyset \wedge L_i \in tlits(C_i), \\ 4 \text{ if } pcgi(A_i \rhd C_i[L_i], \Gamma) = \emptyset \wedge ccgi(A_i \rhd C_i[L_i], \Gamma) \neq \emptyset \wedge L_i \in flits(C_i), \\ 5 \text{ if } pcgi(A_i \rhd C_i[L_i], \Gamma) = \emptyset \wedge ccgi(A_i \rhd C_i[L_i], \Gamma) = \emptyset \wedge L_i \in tlits(C_i), \\ 6 \text{ if } pcgi(A_i \rhd C_i[L_i], \Gamma) = \emptyset \wedge ccgi(A_i \rhd C_i[L_i], \Gamma) = \emptyset \wedge L_i \in flits(C_i), \end{cases}$$

and $V_i(\Gamma) = 4.5$ if $\pi_i(\Gamma) = \Lambda$.

Since a smaller measure is preferable, the empty clause is most preferred; next come clauses with proper constrained ground instances; next come clauses with complementary constrained ground instances; and disposable clauses come last; clauses with I-true selected literal are preferred within each category. Thus, justifications have measure 1; clauses with I-false selected literals contributing proper constrained ground instances to $I^p(\Gamma)$ have measure 2; I-all-true conflict clauses have measure 3; non-I-all-true conflict clauses have measure 4; $V_i(\Gamma) \geq 5$ if and only if $A_i \rhd C_i[L_i]$ is disposable in Γ; and Λ has measure larger than non-disposable clauses and smaller than disposable ones. Then, we define the *measure of a sequence at an index*. Let $\sqcap U$ denote the greatest lower bound of a non-empty set U of ground literals in the SGGS-suitable ordering $\prec$, and let $\sqcap U = M_\infty$, if $U = \emptyset$. For a non-empty set the greatest lower bound exists because $\prec$ is total and well-founded.

Definition 44 (Sequence measure) For $\Gamma = A_1 \rhd C_1[L_1], \ldots, A_n \rhd C_n[L_n]$, the *measure of* Γ *at index* i, for all i, $1 \leq i \leq n$, is the 6-tuple

$$\mathscr{W}(\Gamma, i) = (\sqcap U_i, V_i(\Gamma), |flits(C_i)|, \sqcap X_i, \sqcap Y_i, Gr(A_i \rhd L_i))$$

where $U_i = pcgi(A_i \rhd L_i, \Gamma)$,

- $X_i = \begin{cases} ccgi(A_i \rhd L_i, \Gamma) & \text{if } U_i = \emptyset, \\ \emptyset & \text{otherwise;} \end{cases}$
- $Y_i = \begin{cases} Gr(A_i \rhd L_i) & \text{if } U_i = X_i = \emptyset, \\ \emptyset & \text{otherwise;} \end{cases}$

and $\mathscr{W}(\Gamma, i) = (M_\infty, 4.5, 0, M_\infty, M_\infty, \emptyset)$ if $\pi_i(\Gamma) = \Lambda$.

Thus, if $pcgi(A_i \rhd L_i, \Gamma) \neq \emptyset$, then $\sqcap U_i = pcmin(A_i \rhd L_i, \Gamma)$ and $\sqcap X_i = \sqcap Y_i = M_\infty$; if $pcgi(A_i \rhd L_i, \Gamma) = \emptyset$ and $ccgi(A_i \rhd L_i, \Gamma) \neq \emptyset$, then $\sqcap U_i = M_\infty$, $\sqcap X_i = ccmin(A_i \rhd L_i, \Gamma)$ and $\sqcap Y_i = M_\infty$; if $pcgi(A_i \rhd L_i, \Gamma) = ccgi(A_i \rhd L_i, \Gamma) = \emptyset$, then $\sqcap U_i = \sqcap X_i = M_\infty$ and $\sqcap Y_i = cmin(A_i \rhd L_i)$. If $\pi(\Gamma) = \bot$, then $\mathscr{W}(\Gamma, i) = (M_\infty, 0, 0, M_\infty, M_\infty, \emptyset)$. Let $\preceq_6$ be the quasi-ordering given by the lexicographic combination of $\prec$ for the first component, the ordering $<$ on the non-negative rational numbers for the second component, the ordering $<$ on the natural numbers for the third component, again $\prec$ for the fourth and fifth components, and the subset ordering $\subseteq$ for the sixth component. Note that $\preceq_6$ is a quasi-ordering, that is, it is not strict, because $\subseteq$ is a quasi-ordering. If $\pi_i(\Gamma) = \Lambda$, then $\mathscr{W}(\Gamma, i) \succ_6 \mathscr{W}(\Gamma', i)$, if $\pi_i(\Gamma')$ is not disposable, and $\mathscr{W}(\Gamma, i) \prec_6 \mathscr{W}(\Gamma', i)$, if $\pi_i(\Gamma')$ is disposable. Next, we define *sequence quasi-orderings* that compare two sequences at any index i they both have:

Definition 45 (Sequence quasi-ordering) For all SGGS clause sequences Γ and Γ', for all i, $1 \leq i \leq min(|\Gamma|, |\Gamma'|)$, the *sequence quasi-ordering* $\geq_i$ and its *equivalence relation* $\approx_i$ are defined as follows:

- $\Gamma \geq_i \Gamma'$ if $\mathscr{W}(\Gamma, i) \succeq_6 \mathscr{W}(\Gamma', i)$, and
- $\Gamma \approx_i \Gamma'$ if $\Gamma \geq_i \Gamma'$ and $\Gamma' \geq_i \Gamma$.

As usual, $\Gamma' \leq_i \Gamma$ if $\Gamma \geq_i \Gamma'$; $\Gamma >_i \Gamma'$ if $\Gamma \geq_i \Gamma'$ and $\Gamma' \not\geq_i \Gamma$; and $\Gamma' <_i \Gamma$ if $\Gamma >_i \Gamma'$. We term *sequence ordering* the strict relation $>_i$. The comparison considers first $pcmin(A_i \rhd L_i, \Gamma)$ and $pcmin(A'_i \rhd L'_i, \Gamma')$, and orders them by $\succ$. Since this ordering extends the size ordering, the sequence quasi-orderings favor proper constrained ground instances of smaller size. Intuitively, SGGS tries to build a model using smaller literals first, or, dually, it concentrates on small literals to avoid missing a proof. If $pcmin(A_i \rhd L_i, \Gamma) = pcmin(A'_i \rhd L'_i, \Gamma')$, which includes the case where both are M_∞ because $pcgi(A_i \rhd L_i, \Gamma) = pcgi(A'_i \rhd L'_i, \Gamma') = \emptyset$, then $V_i(\Gamma)$ and $V_i(\Gamma')$ are considered. If Γ and Γ' cannot be ordered at index i by the clause measure, first the numbers of I-false literals in C_i and C'_i are compared, then the minimal complementary constrained ground instances of L_i and L'_i, if they have no proper ones; then the minimal constrained ground instances of L_i and L'_i, if they have neither proper nor complementary ones; and last their sets of constrained ground instances. As expected, the sequence orderings are consistent with disposability:

Lemma 11 *For all SGGS clause sequences Γ and Γ', for all i, $1 \leq i \leq min(|\Gamma|, |\Gamma'|)$, if $\pi_i[\Gamma]$ is disposable in Γ and $\pi_i[\Gamma']$ is not disposable in Γ', then $\Gamma >_i \Gamma'$.*

Proof By hypothesis, $pcgi(\pi_i[\Gamma], \Gamma) = ccgi(\pi_i[\Gamma], \Gamma) = \emptyset$. If $pcgi(\pi_i[\Gamma'], \Gamma') \neq \emptyset$, the claim holds. If $pcgi(\pi_i[\Gamma'], \Gamma') = \emptyset$, the clause measures are compared, and since $V_i(\Gamma) \geq 5$ and $V_i(\Gamma') < 5$, the claim holds. □

For any clause sequence Γ, let Γ^+ denote the sequence $\Gamma\Lambda$. By applying the sequence orderings to prefixes, we obtain an ordering $>^c$ to compare whole sequences:

Definition 46 (Convergence ordering) For all SGGS clause sequences Γ and Γ', the *convergence ordering* $>^c$ and its *equivalence relation* $\approx^c$ are defined as follows:

1. $\Gamma \approx^c \Gamma'$ if $|\Gamma| = |\Gamma'|$ and $\forall i,\ 1 \leq i \leq |\Gamma|+1,\ \Gamma^+ \approx_i (\Gamma')^+$;
2. $\Gamma >^c \Gamma'$ if $\exists i,\ 1 \leq i \leq |\Gamma|+1,\ i \leq |\Gamma'|+1$ such that $\Gamma^+|_{i-1} \approx^c (\Gamma')^+|_{i-1}$ and $\Gamma^+ >_i (\Gamma')^+$.

Furthermore, $\Gamma' <^c \Gamma$ if $\Gamma >^c \Gamma'$. The convergence ordering is basically a lexicographic combination of sequence orderings, where clause sequences are ordered by the smallest index at which they are not equivalent. If a clause sequence is equivalent to a prefix of the other, then Λ makes the difference: the shorter sequence is larger, if the next clause in the longer sequence is not disposable, and it is smaller, if the next clause in the longer sequence is disposable (e.g., $\Gamma\ A \rhd C <^c \Gamma$, if $A \rhd C$ is not disposable, but $\Gamma\ A \rhd C >^c \Gamma$, if $A \rhd C$ is disposable). Intuitively, continuing a sequence with a non-disposable clause may be useful and therefore it makes the longer sequence smaller; continuing it with a disposable clause is useless, and therefore it makes the longer sequence larger. The addition of Λ allows us to achieve this effect while keeping the ordering simple.

The next theorem establishes the well-foundedness of the sequence orderings; since it orders sequences by comparing their measures at a given index i, such an ordering is well-founded on the set of sequences whose length is at least i:

Theorem 6 *For all $i \geq 1$, the sequence ordering $>_i$ is well-founded on the set of SGGS clause sequences $\mathscr{R} = \{\Gamma : |\Gamma| \geq i\}$.*

Proof By way of contradiction, suppose that $\Gamma^1 >_i \Gamma^2 >_i \Gamma^3 \ldots$ is an infinite descending chain of sequences in $\mathscr{R}$. Let the i-th clause in Γ^j be $A_i^j \rhd C_i^j[L_i^j]$. Since the first five components of the quasi-ordering $\succeq_6$ are well-founded, there exists a p, such that for all j, $j \geq p$, the first five components of $\mathscr{W}(\Gamma^j, i)$ are the same as the first five components of $\mathscr{W}(\Gamma^{j+1}, i)$. Let p be the smallest such index. In other words, the first five components of $\mathscr{W}(\Gamma^j, i)$ are constant for all j, $j \geq p$. In the constant tuple made of these first five components, by definition of sequence measure, one of $\sqcap U_i^j$, $\sqcap X_i^j$, and $\sqcap Y_i^j$ is not M_∞. Let L^* be this literal. Because the first five components are constant, in order to have an infinite descending chain, it must be $Gr(A_i^{j+1} \rhd L_i^{j+1}) \subset Gr(A_i^j \rhd L_i^j)$, for all $j \geq p$. It follows that for all j and k such that $j \geq p$, $k \geq p$, and $j \neq k$, literals $A_i^j \rhd L_i^j$ and $A_i^k \rhd L_i^k$ cannot be variants, because if they were, their sets of constrained ground instances would be equal. However, $L_i^j \preceq_s L'$ for all $L' \in Gr(A_i^j \rhd L_i^j)$, and therefore $L_i^j \preceq_s L^*$. In other words, all literals L_i^j for $j \geq p$ are upper bounded in size by L^*, and therefore there cannot be infinitely many of them, excluding variants. Neither can there be infinitely many inequivalent literals $A_i^j \rhd L_i^j$ for finitely many L_i^j. Indeed, for any L_i^j there are finitely many constraints in standard form, as $vars(L_i^j)$ is finite and the signature is finite. By Lemma 10, any constraint that is not in standard form can be transformed into an equivalent constraint that either is in standard form or cannot be modified further by the standardization

rules without violating the upper bound in size. This means that there are finitely many inequivalent constrained literals under a fixed upper bound in size. Thus, the chain $\Gamma^1 >_i \Gamma^2 >_i \Gamma^3 \ldots$ must be finite. □

It follows that a descending chain of sequences of bounded length is finite:

Corollary 2 *If $\Gamma^1 >^c \Gamma^2 >^c \ldots \Gamma^j >^c \Gamma^{j+1} >^c \ldots$, where for all $j \geq 1$, $|\Gamma^j| \leq n$, for some $n \geq 0$, then the chain $\Gamma^1 >^c \Gamma^2 >^c \ldots \Gamma^j >^c \Gamma^{j+1} >^c \ldots$ is finite.*

Proof Since for all $j \geq 1$, $|\Gamma^j| \leq n$, the ordering $>^c$ is a lexicographic combination of orderings $>_i$ for $1 \leq i \leq n$. Since such orderings are well-founded by Theorem 6, $>^c$ is well-founded. □

For most of the following results except refutational completeness, bundledness suffices. For refutational completeness, *fairness* adds a provision which rules out infinite derivations that reduce longer prefixes ignoring the possibility of reducing shorter ones. First, the *index of an inference* is the shortest prefix that an inference can reduce in the convergence ordering:

Definition 47 (Index of an inference) The *index of an SGGS inference $\Gamma \vdash \Gamma'$* is the smallest i such that $\Gamma|_i >^c \Gamma'|_i$ and it is infinite if no such i exists.

Then, the *index of a sequence* is the shortest prefix at which a sequence can be reduced by any applicable inference:

Definition 48 (Index of a sequence) The *index of an SGGS clause sequence Γ*, denoted $index(\Gamma)$, is the minimum index of an SGGS inference applicable to Γ.

Fairness says that any SGGS inference that is infinitely often possible is done eventually:

Definition 49 (Fairness) An SGGS-derivation $\Gamma_0 \vdash \Gamma_1 \vdash \ldots \Gamma_j \vdash \ldots$ is *fair*, if it is bundled, and $\forall i, i > 0$, whenever there are infinitely many Γ_j's such that $index(\Gamma_j) \leq i$, for infinitely many Γ_j's the inference applied to Γ_j has index less than or equal to i.

An SGGS-strategy is *fair* if all its derivations are. For instance, the *minimal index SGGS-strategy* that always selects an inference of minimal index is trivially fair. From now on we consider fair derivations and strategies.

8.2 Completeness

We begin by defining the notion of *limiting sequence*:

Definition 50 (Limiting sequence) An SGGS-derivation $\Gamma_0 \vdash \Gamma_1 \vdash \ldots \vdash \Gamma_j \vdash \ldots$ *admits limit* if there exists an SGGS sequence Γ, such that for all i, $i \leq |\Gamma|$, there is an n_i such that for all j, $j \geq n_i$, if $|\Gamma_j| \geq i$ then $\Gamma_j|_i \approx^c \Gamma|_i$. The longest such sequence, denoted Γ_∞, is the *limiting sequence* of the derivation.

Note that both the derivation and Γ_∞ may be finite or infinite, and if the derivation halts at stage k, Γ_∞ is Γ_k. Corollary 2 applies to sequences of bounded length. The length of sequences in a derivation is not bounded. However, if we consider their prefixes, we have sequences of bounded length. The following theorem applies Corollary 2 to the prefixes of the sequences in an SGGS-derivation:

Theorem 7 *Let $\Gamma_0 \vdash \Gamma_1 \vdash \ldots \vdash \Gamma_j \vdash \ldots$ be an SGGS-derivation. If $\Gamma_1 \geq^c \Gamma_2 \geq^c \ldots \Gamma_j \geq^c \Gamma_{j+1} \geq^c \ldots$, then the derivation admits limit. Furthermore, if its limiting sequence Γ_∞ has finite length, then at most finitely many of the inequalities in the chain are strict.*

Proof By hypothesis, and by the definition of convergence ordering, for all i, $i \geq 0$, $\Gamma_1|_i \geq^c \Gamma_2|_i \geq^c \ldots \Gamma_j|_i \geq^c \Gamma_{j+1}|_i \geq^c \ldots$. Since the $\Gamma_j|_i$'s are sequences of bounded length, by Corollary 2, at most finitely many of these inequalities can be strict. Thus, for all i, there exists an n_i, such that for all $j \geq n_i$, $\Gamma_j|_i \approx^c \Gamma_{j+1}|_i$. Let Γ_∞ be the sequence defined by $\Gamma_\infty|_i = \Gamma_{n_i}|_i$ for all i: Γ_∞ is a limiting sequence. For the second part of the claim, if $|\Gamma_\infty|$ is finite, let N be $\max_<\{n_i : i \leq |\Gamma^\infty|\}$. Then, for all $j \geq N$ and all $i \leq |\Gamma^\infty|$, $\Gamma_j|_i \approx^c \Gamma_\infty|_i$. Thus, for all $j \geq N$, $\Gamma_j \approx^c \Gamma_\infty$. □

The descending chain theorem is a main result that connects the inference rules of the SGGS inference system with the convergence ordering, showing that a fair SGGS-derivation forms a descending chain:

Theorem 8 (Descending Chain Theorem) *If $\Gamma_0 \vdash \Gamma_1 \vdash \ldots \Gamma_j \vdash \ldots$ is a fair SGGS-derivation, then for all j, $j \geq 0$, $\Gamma_j >^c \Gamma_{j+1}$.*

Proof We consider each kind of inference in turn. For all cases, if i is the smallest index where Γ_j and Γ_{j+1} differ, we have $(\Gamma_j)|_{i-1} \approx^c (\Gamma_{j+1})|_{i-1}$ because $(\Gamma_j)|_{i-1} = (\Gamma_{j+1})|_{i-1}$.

1. If $\Gamma_j \vdash \Gamma_{j+1}$ by *SGGS-extension* with extension clause $A \rhd E[L]$, since the derivation is fair, hence bundled, hence sensible, the hypotheses of the lifting theorem holds, and therefore $A \rhd E[L]$ is not disposable by the lifting theorem. If the SGGS-extension is conflicting or non-conflicting non-critical, $\Gamma_{j+1} = \Gamma_j A \rhd E[L]$, and since $A \rhd E[L]$ is not disposable, $\Gamma_j >^c \Gamma_{j+1}$ holds. If the SGGS-extension is critical (cf. Definition 21), $\Gamma_j = \Gamma^1 J \rhd N[O]\Gamma^2$, $\Gamma_{j+1} = \Gamma^1 A \rhd E[L]\Gamma^2$, and $pcmin(A \rhd L, \Gamma^1 A \rhd E[L]) \prec pcmin(J \rhd O, \Gamma^1 J \rhd N[O])$. If i is the index of $J \rhd N[O]$ in Γ_j and of $A \rhd E[L]$ in Γ_{j+1}, we have $\Gamma_j >_i \Gamma_{j+1}$ hence $\Gamma_j >^c \Gamma_{j+1}$.
2. If $\Gamma_j \vdash \Gamma_{j+1}$ is a *partitioning inference*, Γ_j is $\Gamma A \rhd C\langle L\rangle \Gamma'$ and Γ_{j+1} is $\Gamma A_1 \rhd C_1\langle L_1\rangle, \ldots, A_n \rhd C_n\langle L_n\rangle \Gamma'$, for $C_1, \ldots, C_n$ a partition of C. Since the derivation is bundled, $n > 1$, and C is not disposable (if it were, it would be deleted rather than partitioned). Therefore, C has either pcgi's or ccgi's or both. By definition of partitioning inferences (cf. Definitions 23, 24, 27 and 31), C_1 is the preferred clause in the partition. Thus, if $pcgi(C, \Gamma_j) \neq \emptyset$, then $pcgi(C_1, \Gamma_{j+1}) \neq \emptyset$ and $pcmin(L_1, \Gamma_{j+1}) = pcmin(L, \Gamma_j) \neq M_\infty$; if $pcgi(C, \Gamma_j) = \emptyset$ and $ccgi(C, \Gamma_j) \neq \emptyset$, then $pcgi(C_1, \Gamma_{j+1}) = \emptyset$, $ccgi(C_1, \Gamma_{j+1}) \neq \emptyset$, $pcmin(L_1, \Gamma_{j+1}) = pcmin(L, \Gamma_j) = M_\infty$, and $ccmin(L_1, \Gamma_{j+1}) = ccmin(L, \Gamma_j) \neq M_\infty$ $(*)$. Since the L_k's, $1 \leq k \leq n$, are chosen consistently with L, $L_1 \in tlits(C_1)$ if and only if $L \in tlits(C)$ and $L_1 \in flits(C_1)$ if and only if $L \in flits(C)$ $(**)$. Let i be the index of C in Γ_j and of

C_1 in Γ_{j+1}. We compare $\mathscr{W}(\Gamma_j, i)$ and $\mathscr{W}(\Gamma_{j+1}, i)$. For the first component, either $pcmin(L, \Gamma_j) = pcmin(L_1, \Gamma_{j+1}) \neq M_\infty$ or $pcmin(L, \Gamma_j) = pcmin(L_1, \Gamma_{j+1}) = M_\infty$ by $(*)$. For the second component, $V_i(\Gamma_j) = V_i(\Gamma_{j+1})$ by $(*)$ and $(**)$. For the third component, $|flits(C)| \geq |flits(C_1)|$, because an instance of a clause may have fewer, but not more, I-false literals. For the fourth component, by $(*)$ we have either M_∞ on both sides or $ccmin(L, \Gamma_j) = ccmin(L_1, \Gamma_{j+1}) \neq M_\infty$. The fifth component is M_∞ on both sides by $(*)$. For the sixth component, $Gr(A_1 \rhd L_1) \subset Gr(A \rhd L)$, because the partition is not trivial. Altogether we have $\Gamma_j >_i \Gamma_{j+1}$ hence $\Gamma_j >^c \Gamma_{j+1}$.

3. If $\Gamma_j \vdash \Gamma_{j+1}$ by *SGGS-move*, Γ_j is $\Gamma^1 C \Gamma^2 D \Gamma^3$ and Γ_{j+1} is $\Gamma^1 D C \Gamma^2 \Gamma^3$, where C and D abbreviate $A \rhd C[L]$ and $B \rhd D[M]$, $A \rhd C[L]$ is in $dp(\Gamma)$, $B \rhd D[M]$ is I-all-true, and M is assigned to $A \rhd C[L]$, which means L is I-false. Let i be the position of C in Γ_j and of D in Γ_{j+1}. We compare $\mathscr{W}(\Gamma_j, i)$ and $\mathscr{W}(\Gamma_{j+1}, i)$. By Lemma 4, $A \rhd L$ and $B \rhd M$ have the same pcgi's with opposite sign, so that $pcmin(A \rhd L, \Gamma_j) = \neg pcmin(B \rhd M, \Gamma_{j+1})$. Since the ordering $\succ$ ignores sign, the first components of $\mathscr{W}(\Gamma_j, i)$ and $\mathscr{W}(\Gamma_{j+1}, i)$ are equal. For the second component, $V_i(\Gamma_j) = 2$ and $V_i(\Gamma_{j+1}) = 1$, so that $V_i(\Gamma_j) > V_i(\Gamma_{j+1})$, $\Gamma_j >_i \Gamma_{j+1}$ and $\Gamma_j >^c \Gamma_{j+1}$.
4. If $\Gamma_j \vdash \Gamma_{j+1}$ by *SGGS-resolution*, Γ_j has the form $\Gamma^1 D \Gamma^2 C \Gamma^\dagger$ and Γ_{j+1} has the form $\Gamma^1 D \Gamma^2 Res(C, D) \Gamma^{\dagger\dagger}$, where C and D abbreviate $A \rhd C[L]$ and $B \rhd D[M]$, and $Res(C, D) = A \rhd R$ and everything else is as in the definition of SGGS-resolution (cf. Definition 26). Since the derivation is bundled, C is not disposable (if it were, it would be deleted rather than resolved upon). Thus, by Lemma 6, $A \rhd R$ is either $\bot$ or not disposable in Γ_{j+1}. By Lemma 5, $pcgi(A \rhd L, \Gamma_j) = \emptyset$, hence $pcmin(A \rhd L, \Gamma_j) = M_\infty$. Let i be the position of C in Γ_j and of $A \rhd R$ in Γ_{j+1}. We compare $\mathscr{W}(\Gamma_j, i)$ and $\mathscr{W}(\Gamma_{j+1}, i)$. For the first component, assume first that $A \rhd R \neq \bot$ and L_1 is its selected literal. If $pcgi(A \rhd L_1, \Gamma_{j+1}) \neq \emptyset$, $pcmin(A \rhd L, \Gamma_j) \succ pcmin(A \rhd L_1, \Gamma_{j+1})$ and $\Gamma_j >_i \Gamma_{j+1}$. If $pcgi(A \rhd L_1, \Gamma_{j+1}) = \emptyset$ or $A \rhd R = \bot$, we have M_∞ on both sides and we compare $V_i(\Gamma_j)$ and $V_i(\Gamma_{j+1})$. Because C is not disposable in Γ_j, $ccgi(A \rhd L, \Gamma_j) \neq \emptyset$. Since $L \in flits(C)$ (cf. Definition 26), it follows that $V_i(\Gamma_j) = 4$. If $A \rhd R = \bot$, $V_i(\Gamma_{j+1}) = 0$, $V_i(\Gamma_j) > V_i(\Gamma_{j+1})$ and $\Gamma_j >_i \Gamma_{j+1}$. If $A \rhd R \neq \bot$, $A \rhd R$ is not disposable in Γ_{j+1}, and $ccgi(A \rhd L_1, \Gamma_{j+1}) \neq \emptyset$. If $L_1 \in tlits(R)$, then $V_i(\Gamma_{j+1}) = 3$, so that $V_i(\Gamma_j) > V_i(\Gamma_{j+1})$ and $\Gamma_j >_i \Gamma_{j+1}$. If $L_1 \in flits(R)$, then $V_i(\Gamma_j) = V_i(\Gamma_{j+1})$ and we compare $|flits(C)|$ and $|flits(R)|$. Since $B \rhd D[M]$ is I-all-true and $L \in flits(C)$, by construction of R (cf. Definition 26), $|flits(R)| \leq |flits(C)| - 1$ and $\Gamma_j >_i \Gamma_{j+1}$. In all cases we have $\Gamma_j >_i \Gamma_{j+1}$ hence $\Gamma_j >^c \Gamma_{j+1}$.
5. If $\Gamma_j \vdash \Gamma_{j+1}$ by *SGGS-sorting*, Γ_j is $\Gamma^1 C D \Gamma^\dagger$ and Γ_{j+1} is $\Gamma^1 D C \Gamma^{\dagger\dagger}$, where C and D abbreviate $A \rhd C[L]$ and $B \rhd D[M]$, and everything is as in the definition of SGGS-sorting (cf. Definition 29). Let i be the position of C in Γ_j and of D in Γ_{j+1}. By Definition 29, $pcmin(A \rhd L, \Gamma_j) \succ pcmin(B \rhd M, \Gamma_j)$, and both C and D are in $dp(\Gamma_j)$, so that both C and D are in $dp(\Gamma_{j+1})$, and $pcmin(B \rhd M, \Gamma_{j+1}) = pcmin(B \rhd M, \Gamma_j)$, hence $pcmin(A \rhd L, \Gamma_j) \succ pcmin(B \rhd M, \Gamma_{j+1})$. Thus, by the first component of the sequence measure, $\Gamma_j >_i \Gamma_{j+1}$ and $\Gamma_j >^c \Gamma_{j+1}$.
6. If $\Gamma_j \vdash \Gamma_{j+1}$ by *SGGS-deletion*, Γ_{j+1} is Γ_j with all disposable clauses deleted. Let $B \rhd D[M]$, abbreviated D, be the leftmost disposable clause in Γ_j, that is, Γ_j is $\Gamma^1 D \Gamma^2$, where Γ^1 does not contain disposable clauses. Then Γ_{j+1} is $\Gamma^1 \Gamma'$, where Γ' is Γ^2 with all disposable clauses deleted. If Γ' is empty, then $\Gamma_j >^c \Gamma_{j+1}$

by definition of convergence ordering. Otherwise, let i be the index of D in Γ_j, and let $H \rhd N[P]$, abbreviated N, be the clause occurring at index i in Γ_{j+1}, or, equivalently, the leftmost clause in Γ'. Since $H \rhd N[P]$ is not disposable, $\Gamma_j >_i \Gamma_{j+1}$ by Lemma 11, hence $\Gamma_j >^c \Gamma_{j+1}$. □

Since the convergence ordering is well-founded only on SGGS clause sequences of bounded length, the descending chain theorem is perfectly compatible with the semi-decidability of first-order logic: SGGS clause sequences in a derivation are not bounded in length, and infinite derivations with sequences of ever increasing length are possible. The following observations proceed from Theorems 7 and 8:

Corollary 3 *If an SGGS-derivation is fair, then (i) it has limiting sequence, and (ii) every inference has finite index.*

Because SGGS is model-based, model completeness is fairly straightforward:

Theorem 9 (Model Completeness Theorem) *For all input sets S of clauses, initial interpretations I, and fair SGGS-derivations $(S;I;\Gamma_0) \vdash (S;I;\Gamma_1) \vdash \ldots (S;I;\Gamma_j) \vdash \ldots$, if S is satisfiable, then $I[\Gamma_\infty] \models S$.*

Proof By way of contradiction, assume that $I[\Gamma_\infty] \not\models S$. Since the derivation is fair, hence bundled, hence sensible, an SGGS-inference applies to Γ_∞. By Theorem 8, this inference reduces Γ_∞, which is impossible, because it is a limiting sequence.

Refutational completeness requires a preliminary theorem which shows that if a bundled or critical SGGS-extension applies to a sequence, it applies also when the sequence occurs as a prefix of a longer one:

Theorem 10 *If a bundled or critical SGGS-extension derives Γ' from Γ in a fair SGGS-derivation, then for all $\Gamma^\dagger$, the same SGGS-extension derives $\Gamma'\Gamma^\dagger$ from $\Gamma\Gamma^\dagger$.*

Proof Since SGGS-extension derives Γ' from Γ in a fair, hence bundled, hence sensible derivation, $dp(\Gamma) = \Gamma$, which implies that $dp(\Gamma)$ is a prefix of $dp(\Gamma\Gamma^\dagger)$. (If $dp(\Gamma) \neq \Gamma$, then $dp(\Gamma\Gamma^\dagger) = dp(\Gamma)$, so that $dp(\Gamma)$ is a prefix of $dp(\Gamma\Gamma^\dagger)$ anyway.) Thus, if Γ satisfies the requirements of the SGGS-extension inference scheme with side premises $B_1 \rhd D_1[M_1], \ldots, B_n \rhd D_n[M_n]$ and extension clause E, so does $\Gamma\Gamma^\dagger$. Let $\mathscr{I}$ be the set of indices of the side premises in $dp(\Gamma)$.

- If the SGGS-extension is critical, then $\Gamma' = \Gamma^1 E \Gamma^2$, where Γ^1 is the shortest prefix of Γ that enables the critical inference, and $\Gamma^2 \neq \varepsilon$. The same SGGS-extension applied to $\Gamma\Gamma^\dagger$ yields $\Gamma'\Gamma^\dagger$, because if Γ^1 is the shortest prefix of Γ that enables the inference, it is also the shortest such prefix of $\Gamma\Gamma^\dagger$.
- If the SGGS-extension is bundled, its initial conflicting SGGS-extension transforms Γ into ΓE. If E is I-all-true, SGGS-extension with I-all-true conflict clause has no additional requirements on top of those of the SGGS-extension inference scheme. If E is not I-all-true, SGGS-extension with non-I-all-true conflict clause additionally requires that there are clauses $H_1 \rhd F_1[P_1], \ldots, H_k \rhd F_k[P_k]$ in $dp(\Gamma)$ such that $P_1, \ldots, P_k$ are I-true, and the literals in $flits(E)$ depend on the $H_1 \rhd P_1, \ldots, H_k \rhd P_k$. Let $\mathscr{K}$ be the set of indices of the clauses $H_1 \rhd F_1[P_1], \ldots, H_k \rhd$

$F_k[P_k]$ in $dp(\Gamma)$. Since $dp(\Gamma)$ is a prefix of $dp(\Gamma\Gamma^\dagger)$, these clauses occur also in $dp(\Gamma\Gamma^\dagger)$ and at the same indices. Thus, the conflicting SGGS-extension applies to $\Gamma\Gamma^\dagger$ and yields $\Gamma\Gamma^\dagger E$ with the literals in $tlits(E)$ assigned to the indices in $\mathscr{I}$ and the literals in $flits(E)$ depending on the literals at the indices in $\mathscr{K}$. Γ' is the result of applying to ΓE the explanation and solving inferences of the bundled SGGS-extension. By Lemma 9, either Γ' contains $\bot$, or there exists an index i, $i > 0$, such that $\Gamma = \Gamma_1 D \Gamma_2$ with $\pi_i(\Gamma) = D[M]$, $\Gamma' = \Gamma_1 C \Gamma_3$ with $\pi_i(\Gamma') = C$, and $D[M]$ and C are not equivalent. Furthermore, we know from the proof of Lemma 9 that i is the maximum element of the set $\mathscr{I}'$ of the indices which the literals in $tlits(E')$ are assigned to, where E' is E, if E is I-all-true (and in this case $\mathscr{I}' = \mathscr{I}$), or is the result of the explanation inferences for E, if E is not I-all-true. The clauses involved in the explanation inferences for E are precisely those at the indices in $\mathscr{K}$. Since E, $\mathscr{K}$, and $\mathscr{I}$ are the same regardless of whether the SGGS-extension is applied to Γ or $\Gamma\Gamma^\dagger$, it follows that the explanation inferences, E', $\mathscr{I}'$, i, and the solving inferences are also the same. Therefore, the result of the explanation and solving inferences applied to $\Gamma\Gamma^\dagger E$ is $\Gamma'\Gamma^\dagger$. □

Since in a fair derivation all conflicting SGGS-extensions are bundled, Theorem 10 covers all SGGS-extensions except non-conflicting non-critical ones. In the next theorem a non-conflicting non-critical SGGS-extension that applies to a prefix is covered by a critical SGGS-extension that applies to the whole sequence:

Theorem 11 (Refutational Completeness Theorem) *For all input sets S of clauses and initial interpretations I, if S is unsatisfiable, any fair SGGS-derivation* $(S;I;\Gamma_0) \vdash (S;I;\Gamma_1) \vdash \ldots (S;I;\Gamma_j) \vdash \ldots$ *is a refutation.*

Proof By way of contradiction, assume that for all $j \geq 0$, Γ_j does not contain $\bot$. Because the derivation is fair, hence bundled, hence sensible, SGGS-deletion is applied eagerly, and therefore without loss of generality we can consider any Γ_j without disposable clauses. For all $j \geq 0$, since S is unsatisfiable, $I[\Gamma_j] \not\models S$. Since the derivation is sensible, if $\Gamma_j \neq dp(\Gamma_j)$, an inference other than SGGS-extension applies to Γ_j; and if $\Gamma_j = dp(\Gamma_j)$, an SGGS-extension applies to Γ_j. It follows that the derivation is infinite, and, by the descending chain theorem, it forms a chain $\Gamma_0 >^c \Gamma_1 >^c \ldots \Gamma_j >^c \ldots$. By Theorem 7, the derivation admits limit, and its limiting sequence Γ_∞ is infinite, because the derivation is a chain that features infinitely many strict inequalities. Let Γ_∞ be $C_1[L_1], \ldots, C_n[L_n], \ldots$, where we omit the constraints for brevity. The selected literals L_i's of Γ_∞ must all be disjoint, because otherwise, by Lemma 8, an SGGS inference would apply and decrease Γ_∞ by Theorem 8, contradicting the fact that Γ_∞ is the limiting sequence. This implies that $\Gamma_\infty = dp(\Gamma_\infty)$ and we have infinitely many disjoint literals $L_1, \ldots, L_n, \ldots$. The latter implies that the L_i's become arbitrarily large in the size ordering and therefore in the SGGS-suitable ordering $\succ$.
Let S' be a finite unsatisfiable set of ground instances of clauses in S. Such a set exists by Herbrand's theorem because S is unsatisfiable. Let Q be the largest literal in S' according to the ordering $\succ$. Say that a literal L is *small*, if $L \preceq Q$, and *large* otherwise. Because the L_i's are disjoint, there are only finitely many L_i's that are small. Let r, where $r \geq 1$, be the largest index in Γ_∞, such that the selected literal L_r is small. Let Γ^{small} be $(\Gamma_\infty)|_r$: all selected literals of Γ_∞ that are small occur in Γ^{small}, but also large

selected literals may be in Γ^{small}. Let C^* be a clause in S' such that $I[\Gamma_\infty] \not\models C^*$. Such a clause exists because S' is unsatisfiable. We show that $I[\Gamma^{small}] \not\models C^*$, because all literals in C^* are small. Indeed, first we observe that if a literal L is small and depends on a literal M, then M also is small. Then we consider that $I[\Gamma_\infty] \not\models C^*$ means that for all literals L in C^*, either L is I-true and depends on an I-false selected literal M in Γ_∞, or L is I-false and depends on an I-true selected literal M in Γ_∞, or L is I-false and $at(L) \notin at(I^p(\Gamma_\infty))$. In the first two cases, M must be small and therefore it is in Γ^{small}; in the third case, $at(L) \notin at(I^p(\Gamma_\infty))$ implies $at(L) \notin at(I^p(\Gamma^{small}))$. Thus, $I[\Gamma^{small}] \not\models C^*$.

Let C be a clause in S such that C^* is instance of C: since all literals in C^* are small and $\succ$ extends the size ordering, it follows that all literals in C are small. Let j be the smallest index in the derivation such that $(\Gamma_j)|_n = (\Gamma_\infty)|_n$, for some n such that $n > r$ and $(\Gamma_j)|_n$ is the longest prefix of Γ_j for which $(\Gamma_j)|_n = (\Gamma_\infty)|_n$. Then Γ_j has the form $C_1[L_1], \ldots, C_n[L_n], D_1[M_1], \ldots, D_k[M_k]$, for some $k > 0$, where $C_1, \ldots, C_n$ is $(\Gamma_j)|_n = (\Gamma_\infty)|_n$, but $C_1, \ldots, C_n, D_1$ is not a prefix of Γ_∞, L_r is small, $L_{r+1}, \ldots L_n$ are all large, and $L_1, \ldots, L_{r-1}, M_1, \ldots M_k$ can be either small or large. From $\Gamma_\infty = dp(\Gamma_\infty)$ and $(\Gamma_j)|_n = (\Gamma_\infty)|_n$, it follows that $(\Gamma_j)|_n = dp((\Gamma_j)|_n)$. Since all literals of C^* are small, and $I[\Gamma^{small}] \not\models C^*$, it follows that $I[(\Gamma_j)|_n] \not\models C^*$, because $I[(\Gamma_j)|_n]$ may differ from $I[\Gamma^{small}]$ only on large literals. By the lifting theorem, there is a constrained clause $A \rhd E[L]$ such that E is an instance of C, C^* is a constrained ground instance of $A \rhd E[L]$, and $A \rhd E[L]$ can be added to $(\Gamma_j)|_n$ by SGGS-extension. By applying to E and C^* the same reasoning applied above to C and C^*, we have that all literals in E are small. If the SGGS-extension of $(\Gamma_j)|_n$ with $A \rhd E[L]$ is bundled or critical, by Theorem 10 it applies also to Γ_j.

Otherwise, it is a non-conflicting non-critical SGGS-extension. We show that a critical SGGS-extension extends Γ_j with $A \rhd E[L]$, by showing that Γ_j has the form $\Gamma^1 N[O] \Gamma^2$, required by a critical SGGS-extension (cf. Definition 21), with Γ^1 given by Γ^{small}. First, since the SGGS-extension of $(\Gamma_j)|_n$ with $A \rhd E[L]$ is non-conflicting non-critical, $A \rhd E[L]$ is not I-all-true, and $at(Gr(A \rhd L)) \not\subseteq at(pcgi(H \rhd P, \Gamma))$ for all selected literals $H \rhd P$ in $(\Gamma_j)|_n$. It follows that $at(Gr(A \rhd L)) \not\subseteq at(pcgi(H \rhd P, \Gamma))$ for all selected literals $H \rhd P$ in Γ^{small}. Second, Γ^{small} contains all side premises of the inference: indeed, the I-true literals of E, that are small because all literals of E are, depend on the selected literals of the side premises: this means that the selected literals of the side premises are also small, and therefore must be in Γ^{small}. Third, Γ^{small} is the shortest prefix enabling the inference, by the way r is defined (e.g., a shorter prefix may not contain all side premises). Fourth, $N[O]$ is $C_{r+1}[L_{r+1}]$, for which $pcmin(L, \Gamma^1 E[L]) \prec pcmin(L_{r+1}, \Gamma^1 C_{r+1}[L_{r+1}])$, because L_{r+1} is large, whereas L is small. Fifth, Γ^2 is $C_{r+2}[L_{r+2}], \ldots, C_n[L_n], D_1[M_1], \ldots, D_k[M_k]$, where $\Gamma^2 \neq \varepsilon$, because $n > r > 1$.

Thus, in all cases, SGGS-extension with $A \rhd E[L]$ applies to Γ_j. Since j is the smallest index such that $(\Gamma_j)|_n = (\Gamma_\infty)|_n$, there are infinitely many Γ_j's such that $index(\Gamma_j) \leq n+1$, and therefore by fairness the inference applied to Γ_j has index less than or equal to $n+1$. By Theorem 8, SGGS-extension with $A \rhd E[L]$ decreases strictly $(\Gamma_j)|_n$ in the convergence ordering, contradicting the fact that $(\Gamma_j)|_n = (\Gamma_\infty)|_n$ and $(\Gamma_\infty)|_n$, being a prefix of the limiting sequence, cannot be reduced. □

This proof elucidates the rôle of critical SGGS-extensions: they are needed to avoid overlooking proofs made of "small" ground instances in a system that does not enumerate ground instances.

8.3 Goal Sensitivity

An input set of clauses S is typically generated from transforming into clausal form a set of formulae, considered as assumptions, and the negation $\neg\varphi$ of a formula φ considered as conjecture. A theorem-proving method is *goal-sensitive*, if it only performs inferences that involve clauses in, or deduced from, the clausal form of $\neg\varphi$. Let $S = T \uplus iSOS$, where $iSOS$ is the set of clauses obtained from the transformation into clausal form of $\neg\varphi$ and T is its complement. The acronym $iSOS$ stands for *input set of support*, which is the traditional name of this subset of clauses in theorem proving. We show that SGGS is goal-sensitive, if I is properly chosen:

Definition 51 (Goal sensitivity) An initial interpretation I is *goal-sensitive* for input set $S = T \uplus iSOS$, if $I \models T$ and $I \not\models iSOS$.

If T is consistent, and S is unsatisfiable, goal-sensitive interpretations are guaranteed to exist.

Definition 52 (Ground link) Two ground clauses C and D are *linked*, denoted $C \rightleftharpoons D$, if there exist literals L_1 of C and L_2 of D, such that $L_1 = \neg L_2$.

Let $\rightleftharpoons^*$ be the reflexive transitive closure of $\rightleftharpoons$. Ground clauses C and D are *connected* if $C \rightleftharpoons^* D$. Let $Gr(S) = \{C' : C' \in Gr(C),\ C \in S\}$.

Definition 53 (Closure) Given a set of clauses S, a set $\mathscr{G}$ of ground clauses is *connection-closed* with respect to S, if for all $C \in \mathscr{G}$ and $D \in Gr(S)$, $C \rightleftharpoons^* D$ implies $D \in \mathscr{G}$; it is *closed with respect to resolution*, if $C, D \in \mathscr{G}$ implies $R \in \mathscr{G}$ for all resolvents R of C and D.

For an SGGS clause sequence Γ of length n, let $Gr(\Gamma) = \bigcup_{i=1}^{n} Gr(\pi_i(\Gamma))$.

Definition 54 (Goal-relevance) Given input set $S = T \uplus iSOS$, the set $\mathscr{G}_S$ of *goal-relevant* clauses is the $\subseteq$-smallest set of ground clauses such that (i) for all $C \in iSOS$, $Gr(C) \subseteq \mathscr{G}_S$, (ii) $\mathscr{G}_S$ is connection-closed with respect to S, and (iii) $\mathscr{G}_S$ is closed with respect to resolution.

A constrained clause $A \rhd C$ is *goal-relevant* for S, if $Gr(A \rhd C) \subseteq \mathscr{G}_S$. An SGGS clause sequence Γ is *goal-relevant* for S, if $Gr(\Gamma) \subseteq \mathscr{G}_S$. An SGGS-derivation $\Gamma_0 \vdash \Gamma_1 \vdash \ldots \Gamma_j \vdash \ldots$ from S is *goal-sensitive*, if $\forall j,\ j \geq 0$, Γ_j is goal-relevant for S.

Theorem 12 (Goal-sensitivity Theorem) *For all input sets S of clauses and initial interpretations I, if I is goal-sensitive, any SGGS-derivation $(S;I;\Gamma_0) \vdash (S;I;\Gamma_1) \vdash \ldots (S;I;\Gamma_j) \vdash \ldots$ is goal-sensitive.*

Proof The proof is by induction on the length of the derivation. Base case: $\Gamma_0 = \varepsilon$ is vacuously goal-relevant for S. Induction hypothesis: Γ_n is goal-relevant for S. Induction step: we consider all possible SGGS inferences $\Gamma_n \vdash \Gamma_{n+1}$:

- If $\Gamma_n \vdash \Gamma_{n+1}$ is an SGGS-deletion or SGGS-sorting step, goal-relevance is trivially unaffected.
- If $\Gamma_n \vdash \Gamma_{n+1}$ is a partitioning inference, it replaces a clause C by a partition $C_1, \ldots, C_n$, where $Gr(C_i) \subseteq Gr(C)$ for all i, $1 \leq i \leq n$. Since by induction hypothesis $Gr(C) \subseteq \mathscr{G}_S$, it follows that $Gr(C_i) \subseteq \mathscr{G}_S$ for all i, $1 \leq i \leq n$.
- For SGGS-resolution, we observe that all cgi's of a resolvent are resolvents of cgi's of the parents. Consider clauses $C \vee L$ and $D \vee M$, where C and D are disjunctions of literals, and $L\vartheta = \neg M\vartheta$ with most general unifier ϑ, so that the resolvent is $R = (C \vee D)\vartheta$. For all $R\alpha \in Gr(R)$, $R\alpha$ is a resolvent of $(C \vee L)\vartheta\alpha' \in Gr(C \vee L)$ and $(D \vee M)\vartheta\alpha' \in Gr(D \vee M)$, where α' is $\alpha \cup \sigma$, and σ is a substitution that replaces with ground terms all variables in $(vars(L) \setminus vars(C)) \cup (vars(M) \setminus vars(D))$, if any, and is empty otherwise. If $\Gamma_n \vdash \Gamma_{n+1}$ is an SGGS-resolution step with parents C and D and resolvent R, by induction hypothesis $Gr(C) \subseteq \mathscr{G}_S$ and $Gr(D) \subseteq \mathscr{G}_S$, and $Gr(R) \subseteq \mathscr{G}_S$ follows, because $\mathscr{G}_S$ is closed with respect to resolution.
- If $\Gamma_n \vdash \Gamma_{n+1}$ is an SGGS-extension step with extension clause E, we distinguish two cases. If $tlits(E) = \emptyset$, E is an I-all-false instance of a clause in S. Since I is goal-sensitive by hypothesis, no instance of a clause in T can be I-all-false. Thus, E is an instance of a clause in *iSOS*, and therefore $Gr(E) \subseteq \mathscr{G}_S$. If $tlits(E) \neq \emptyset$, every $L \in tlits(E)$ is assigned to a side premise $D[M]$ and depends on M, so that every cgi of E is connected to a cgi of D. Since the side premises are clauses of Γ_n, by induction hypothesis, $Gr(D) \subseteq \mathscr{G}_S$. Since E is an instance of a clause in S, $Gr(E) \subseteq Gr(S)$. Since $\mathscr{G}_S$ is connection-closed with respect to S, $Gr(E) \subseteq \mathscr{G}_S$ also holds. □

In practice, identifying *iSOS* with the clausal form of a negated conjecture may not be always helpful or even possible. For example, if S is a knowledge base and the problem is to detect inconsistencies, goal sensitivity may still be useful, by taking as T the largest consistent subset of S and as *iSOS* its complement $S \setminus T$.

9 Discussion

9.1 Related work

Since SGGS-extension generates instances of input clauses, SGGS is related to *instance-based reasoning* [12,4], the theorem-proving paradigm most directly inspired by Herbrand's theorem. The basic idea is to generate ground instances of input clauses, and test them for propositional unsatisfiability. The first such procedure was *Gilmore method* (e.g., [19]), followed much later by *SATCHMO* [47], and *hyper-linking* [39], the latter at the beginning of the renewed interest for the DPLL procedure [26,25,19] for propositional satisfiability. Proceeding instance-based strategies, such as *CLINS-S* [20,21], *ordered semantic hyperlinking* (OSHL) [55,54], *Inst-Gen* [30,31,37,36],

as well as methods that hybridize instance generation and tableaux, such as the *disconnection calculus* [11,40–42] and *hypertableaux* [2,10], progressively emphasized *model-driven* instance generation, where the procedure maintains a candidate model, generates ground instances to exclude it, updates it to satisfy them, and continues until a contradiction arises. Surveys on instance-based theorem proving appeared in [33,35], where these systems are classified depending on whether the interleaving of instance generation and ground reasoning is *coarse* or *fine* grained. While SGGS develops the idea of model-driven instance generation issued from this line of research, both its inference system and its model representation approach are different from all their predecessors. The instances generated by SGGS-extension are not ground in general, and SGGS does not interleave instance generation and propositional or ground reasoning.

Several methods generalize features of DPLL to first-order logic. One thread of research [50,61,57,29,32,28] pursued endowing resolution and superposition with a *first-order splitting* mechanism, which decomposes a clause into subdisjunctions that do not share variables: for example, clause $A(x) \vee B(y)$ yields two cases, one with $A(x)$ and one with $B(y)$, to be explored via backtracking. A motivation was to improve the performance of resolution and superposition on long non-Horn clauses. First-order splitting was inspired by viewing DPLL splitting as clause decomposition, rather than as a guess of a truth value as in CDCL decision, since resolution and superposition are not model-based.

Clearly, one cannot decompose as above $A(x) \vee B(x)$, because there is a variable in common. Breaking clauses apart is a native feature of tableaux, whose well-known downside is given by *rigid variables*: the procedure has to use backtracking to undo substitutions global to the tableau. Thus, backtracking is used to go from a tableau to another tableau, rather than from a branch to another branch of a tree as in DPLL. The comparison between resolution (proof confluent) and tableaux (not proof confluent) led to define proof confluence as used in this article (cf. Section 8 of [18] for a discussion). An approach to this problem was offered by the already mentioned *disconnection* [11,40–42] and *hypertableaux* [2,10] calculi, which refrain from instantiating tableau variables and add instances to the set of clauses.

A different approach was suggested in *FDPLL* [3] and developed in the *model-evolution* calculus [5,6,8,9,7]: the candidate model is represented by a sequence of first-order literals; splitting $A(x) \vee B(x)$ yields a branch with $A(Y)$ and one with $\neg A(Y)$, where Y is a *parameter*, as parameters are introduced to avoid as much as possible using Skolem constants. Model evolution is considered a faithful lifting of DPLL to first-order logic, originally motivated by lifting splitting. SGGS lifts CDCL to first-order logic. DPLL [26,25,19] and CDCL [49,51,48,46] are regarded as two distinct methods,[1] and both model representation and inferences in SGGS differ from those of model evolution.

Another research line investigated pipelining [13] or integrating [27,15] resolution and superposition based engines with solvers for *satisfiability modulo theories*,

[1] An early forerunner of CDCL was the *semantic tree method with lemma generation* (cf. pages 284–288 in [53]).

that build theories on top of a CDCL-based SAT-solver. The AVATAR architecture [56] inherits from both first-order splitting [57,32] and integration as in [27,15].

DPLL($\mathscr{S}\mathscr{X}$) [52] and NRCL [1] generalize CDCL to *effectively propositional logic* (EPR). This fragment of first-order logic, also known as the *Bernays-Schönfinkel class*, allows only formulæ of the form $\exists^*\forall^*\varphi$, where φ is quantifier-free and function-free, while constant symbols are permitted. EPR satisfiability reduces to SAT by replacing existentially quantified variables by Skolem constants, replacing universally quantified variables by constants in all possible ways, and abstracting ground first-order atoms to propositional atoms. DPLL($\mathscr{S}\mathscr{X}$) avoids exhaustive grounding by decorating EPR clauses with sets of substitutions, that are manipulated by operations similar to those of relational algebra. CDCL is lifted to EPR by adding factoring and making all CDCL operations aware of the substitution sets. NRCL, that stands for *Non-Redundant Clause Learning*, employs a sequence of constrained EPR literals to represent a candidate partial model, and guarantees that no learned conflict clause is redundant. Also SGGS has this property, since a clause that enters the disjoint prefix with an SGGS-move is the justification of its selected literal, and therefore it is not disposable. SGGS recreates CDCL in a semi-decision procedure for full first-order logic, handling also possibly infinite Herbrand bases.

For the part on constraints, we recall that an *equational problem* is a first-order formula with equality as the only predicate symbol. Equalities between terms interpreted in the Herbrand universe are known as *Herbrand constraints*. Solved forms for equational problems, inference rules for reduction to solved form, and their termination were investigated independently in [44,45,23], yielding procedures to decide the validity of an equational problem in the Herbrand universe. The work in [44,45,23] was motivated primarily by logic programming; a survey of this kind of research was given in [22]. *SGGS constraints* are a variant of Herbrand constraints: by featuring atomic constraints in the form $top(t) = f$, they allow SGGS to avoid quantifiers in constraints. Indeed, $top(t) = f$ replaces $\exists x_1 \dots \exists x_n.\ t \equiv f(x_1,\dots,x_n)$, and $top(t) \neq f$ replaces $\forall x_1 \dots \forall x_n.\ t \not\equiv f(x_1,\dots,x_n)$. As a consequence, our notion of standard form is different from the solved forms studied in [23] for equational problems, that feature quantifiers. Our inference rules for constraints, and our results on the termination of their application, are tailored for SGGS. The interested reader may find additional material on related work in [14,18].

9.2 Summary of contributions and future work

We presented and proved refutationally complete a new theorem-proving method, called SGGS, that lifts to first-order logic the *conflict-driven clause learning* (CDCL) procedure of propositional SAT solving [49,51,48,46]. SGGS has the possibly unique property of being simultaneously *model-based*, *semantically guided*, *goal-sensitive*, and *proof confluent*. It searches for a model of the input set S of clauses, starting from a given *initial interpretation* $I = I[\Gamma_0]$, and building interpretations $I[\Gamma_1]$, $I[\Gamma_2]$, $I[\Gamma_3]\dots$, represented by *SGGS clause sequences*, or sequences of constrained clauses with selected literals, that make an SGGS-derivation $\Gamma_0 \vdash \Gamma_1 \vdash \Gamma_2 \vdash \Gamma_3 \vdash \dots$.

The main operations of CDCL are decision, Boolean clausal propagation, conflict explanation by propositional resolution, learning a conflict clause, and conflict solving by backjumping. The analogue of decision in SGGS is *selection* of a literal in every clause in an SGGS clause sequence Γ, since selected literals differentiate $I[\Gamma]$ from Γ. *First-order clausal propagation* in SGGS relies on the concepts of *uniform falsity* and *dependence*. A literal is *uniformly false* in an interpretation, if all its ground instances are false in that interpretation; for I, a literal is I-true if it is true in I, and I-false if it is uniformly false in I. A literal L *depends* on a selected literal M, if M precedes L in Γ, and all ground instances of L appear negated among the *proper* constrained ground instances of M, or those that M contributes to $I[\Gamma]$, so that M's selection makes L *uniformly false* in $I[\Gamma]$.

All SGGS operations work *modulo semantic guidance* by I, because the system endeavours to make $I[\Gamma]$ different from I, since $I \not\models S$ (otherwise, the problem is solved). Thus, it is the I-false selected literals in Γ that differentiate $I[\Gamma]$ from I; it is the dependence of I-true literals on I-false selected literals that is recorded by *assignments*; and it is I-all-true clauses, or clauses whose literals are all I-true, that are *conflict clauses* or *justifications* of *implied literal*. When all literals of an I-all-true clause are assigned, it means that in an attempt to diversify $I[\Gamma]$ from I to satisfy other clauses, the system made that I-all-true clause uniformly false in $I[\Gamma]$. When all literals of an I-all-true clause but one are assigned, the non-assigned one must be selected, and it is an implied literal, as *all* its ground instances must be true in $I[\Gamma]$ to satisfy the clause. Clauses whose selected literals contribute all their ground instances to $I[\Gamma]$ form the *disjoint prefix* $dp(\Gamma)$, that is, the best part of Γ.

The SGGS inference system includes *SGGS-extension*, *SGGS-splitting*, *SGGS-resolution*, *SGGS-move*, and *SGGS-deletion*. We gave *SGGS-extension* and *SGGS-splitting* as *inference schemes*, each instantiated into four inference rules. SGGS-extension is an *instance generation* mechanism, and it is a hyperinference, like *hyperresolution* [58], *hyperlinking* [39], and *hypertableaux extension* [2], in that it applies a simultaneous most general unifier of possibly multiple pairs of literals from an input clause and the current Γ. It further instantiates the clause with a new kind of substitution, called *most general semantic falsifier*, to maintain the invariant that all literals in Γ are either I-true or I-false. This invariant ensures that all ground instances of a literal in an SGGS clause sequence are in harmony with respect to I. Most general semantic falsifiers are empty if I is all-positive or all-negative.

SGGS-splitting of clause C by clause D replaces C by a *partition*, where all ground instances that a specified literal in C has in common with D's selected literal are confined to one element. This enables SGGS-resolution or SGGS-deletion to remove such an intersection, ridding the candidate model of duplications or contradictions. *SGGS-resolution* is a restricted form of first-order resolution, where an implied literal in a justification resolves away a literal that depends on it: for this reason it uses *matching* rather than unification, and allows the resolvent to *replace* the non-I-all-true parent. *SGGS-deletion* removes *disposable* clauses that are satisfied by the interpretation induced by the clauses on their left in Γ.

SGGS-extension extends the sequence Γ and the candidate model $I[\Gamma]$, while the other inference rules amend or shrink them, so that all inferences are *model-based*. This behavior is captured by the *lifting theorem* (cf. Theorem 4), which shows

that if $I[\Gamma] \not\models S$ and $dp(\Gamma) = \Gamma$, SGGS-extension applies, and its companion (cf. Theorem 3), which shows that if $dp(\Gamma) \neq \Gamma$, another inference rule applies.

If SGGS-extension adds a clause in conflict with $I[\Gamma]$ the first-order CDCL mechanism of SGGS is brought to bear. It comprises *explanation* and *solving* inferences. If the conflict clause includes I-false literals, SGGS-resolution *explains* the conflict by resolving away those I-false literals with implied literals in $dp(\Gamma)$. An SGGS-extension adding such a clause makes sure that this is possible by applying an *extension substitution*. The explanation inferences yield either the empty clause or an I-all-true conflict clause, which is then subject to the solving inferences. If the conflict clause does not include I-false literals, only the solving inferences are employed: the conflict clause is moved to the left of the clause which its selected literal is assigned to. This *SGGS-move* solves the conflict by *flipping* the truth value in $I[\Gamma]$ of *all* ground instances of this literal. It corresponds to backjumping from a branch of the tree to another one in CDCL. The candidate model is thus amended with no need to undo steps and return to previous states, so that SGGS is *proof confluent*. The moved clause is learned in the sense that it enters $dp(\Gamma)$ as justification of its selected literal. Prior to the move, splitting inferences may apply to make the selected literal of the clause to be moved so precise, that the move will indeed flip the truth value of all its ground instances. Every SGGS-extension with a conflict clause is *bundled* with the explanation and solving inference that solve the conflict.

Refutational completeness is established by exhibiting a *convergence ordering* on SGGS clause sequences, that is *well-founded* on sequences of *bounded length*. The length of sequences in a first-order SGGS-derivation is *not* bounded. However, this well-foundedness property allows us to compare the prefixes of the sequences in a derivation and define its *limiting sequence*: it is the longest sequence Γ_∞ such that for all lengths i, $i \leq |\Gamma_\infty|$, there exists a stage of the derivation beyond which the prefix of length i of every sequence in the derivation is equivalent to that of Γ_∞. We proved a *descending chain theorem* (cf. Theorem 8) which shows that SGGS inferences reduce SGGS clause sequences in the convergence ordering. It follows that any SGGS-derivation admits limiting sequence Γ_∞ (cf. Theorem 7), and if the input set S is satisfiable, $I[\Gamma_\infty]$ is a model of S (cf. Theorem 9). The refutational completeness theorem (cf. Theorem 11) shows that whenever the input set is unsatisfiable, any fair SGGS-derivation is guaranteed to generate the empty clause: otherwise, we would have an infinite derivation with infinite Γ_∞, and SGGS inferences that reduce a prefix of Γ_∞, contradicting that it is the limiting sequence.

The inception of SGGS opens many directions for future research. The first one is to implement SGGS, which involves designing algorithms to compute efficiently its operations, and devising strategies for efficient inference control. Literal selection and literal assignment (cf. Definition 9) are examples of heuristic choice. Heuristics for literal selection in SGGS may take inspiration from *activity-based* heuristics for decision in CDCL, that count how many times a propositional variable is resolved upon and favor most active variables. Performance evaluation of an implementation can be complemented with complexity analyses of SGGS actions, such as splitting a clause by another, and, for an I not based on sign, computing most general semantic falsifiers and checking that literals are I-false or I-true. Feed-back from an implementation may lead to simplify SGGS, and a simpler or more abstract presentation

may be sought also at the theoretical level. *Redundancy criteria* and *contraction inference rules* that delete or replace redundant clauses are well-known to be of crucial importance for first-order theorem proving. They are typically based on well-founded orderings (e.g., the subsumption ordering). SGGS has a *model-based* redundancy criterion, namely *disposability*, which includes forward subsumption. SGGS also features a well-founded total ordering on ground literals, that is used to replace a clause by another one in critical SGGS-extensions, and to sort clauses by SGGS-sorting. In the context of a more abstract study of SGGS, model-based and ordering-based redundancy could be unified. SGGS may also delete some of the clauses produced by SGGS-splitting, as well as clauses including literals assigned to a partitioned clause or to the non-I-all-true parent of an SGGS-resolution step. While such deletions resemble a partial *restart* in CDCL parlance, it may be possible to cover them also in a study of redundancy for model-based reasoning.

Further topics include extending SGGS with theory reasoning, beginning with equality, and determining whether it is a decision procedure for decidable fragments. In its search for a model of the input set, SGGS guesses I-false literals and learns I-true implied literals: if I is based on sign (e.g., all-negative), this means guessing literals of opposite sign (e.g., positive) and learning literals of the same sign (e.g., negative). A major issue is how to give initial interpretations not based on sign. This theme is connected with theory reasoning, because a way to capture an interpretation is to say that it is a model of a theory, either presented by a set of axioms, or built into a solver that constructs a model and acts as oracle for truth in that model. Examples include fragments of set theory or lattice theory, fragments of algebra, Boolean logic, implicational logic, and theories of data structures such as lists. While state-of-the-art theorem provers are very sophisticated (e.g., [5, 62, 37, 38, 59]), SGGS propounds a semantically-oriented style of theorem proving, that may be rewarding for hard problems or new domains, whose identification is also an objective. If SGGS had on first-order theorem proving an effect even partially similar to that of CDCL on SAT, the impact could be far-reaching.

Acknowledgements An abstract of an early version of this work was presented at the Meeting of the IFIP Working Group 1.6 on Rewriting at the Sixth Federated Logic Conference (FLoC) at the Vienna Summer of Logic in July 2014. This article was completed when the first author was an international fellow at the Computer Science Laboratory of SRI International in Menlo Park, and also during a visit at Microsoft Research in Redmond: the support of both institutions is gratefully acknowledged. We thank the reviewers for their comments that helped us improve our manuscript.

References

1. Gábor Alagi and Christoph Weidenbach. NRCL – a model building approach to the Bernays-Schönfinkel fragment. In Carsten Lutz and Silvio Ranise, editors, *Proceedings of the 10th International Symposium on Frontiers of Combining Systems (FroCoS)*, volume 9322 of *Lecture Notes in Artificial Intelligence*, pages 69–84, Berlin, 2015. Springer.
2. 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, Berlin, 1998. Springer.

3. 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, Berlin, 2000. Springer.
4. Peter Baumgartner. Logical engineering with instance-based methods. In Frank Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction (CADE)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 404–409, Berlin, 2007. Springer.
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, Berlin, 2006. Springer.
7. Peter Baumgartner, Björn Pelzer, and Cesare Tinelli. Model evolution calculus with equality – revised and implemented. *Journal of Symbolic Computation*, 47(9):1011–1045, 2012.
8. Peter Baumgartner and Cesare Tinelli. The model evolution calculus as a first-order DPLL method. *Artificial Intelligence*, 172(4–5):591–632, 2008.
9. 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, Berlin, 2009. Springer.
10. 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, Berlin, 2013. Springer.
11. 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, Berlin, 1996. Springer.
12. 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, Berlin, 1999.
13. Maria Paola Bonacina and Mnacho Echenim. Theory decision by decomposition. *Journal of Symbolic Computation*, 45(2):229–260, 2010.
14. Maria Paola Bonacina, Ulrich Furbach, and Viorica Sofronie-Stokkermans. On first-order model-based reasoning. In Narciso Martí-Oliet, Peter Olveczky, and Carolyn Talcott, editors, *Logic, Rewriting, and Concurrency: Essays Dedicated to José Meseguer*, volume 9200 of *Lecture Notes in Computer Science*, pages 181–204. Springer, Berlin, 2015.
15. 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.
16. 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`.
17. Maria Paola Bonacina and David A. Plaisted. SGGS theorem proving: an exposition. In Stephan Schulz, Leonardo De Moura, and Boris Konev, editors, *Proceedings of the 4th Workshop on Practical Aspects in Automated Reasoning (PAAR) (2014)*, volume 31 of *EasyChair Proceedings in Computing (EPiC)*, pages 25–38, July 2015.
18. Maria Paola Bonacina and David A. Plaisted. Semantically-guided goal-sensitive reasoning: model representation. *Journal of Automated Reasoning*, 56(2):113–141, 2016.
19. Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Cambridge, 1973.
20. Heng Chu and David A. Plaisted. Model finding in semantically guided instance-based theorem proving. *Fundamenta Informaticae*, 21(3):221–235, 1994.
21. Heng Chu and David A. Plaisted. CLINS-S: a semantically guided first-order theorem prover. *Journal of Automated Reasoning*, 18(2):183–188, 1997.
22. Hubert Comon. Disunification: a survey. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic – Essays in Honor of Alan Robinson*, pages 322–359. The MIT Press, Cambridge, 1991.

23. Hubert Comon and Pierre Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7:371–425, 1989.
24. 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.
25. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.
26. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
27. 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, Berlin, 2008. Springer.
28. Arnaud Fietzke. *Labelled superposition*. PhD thesis, Max Planck Institut für Informatik, Saabrücken, October 2013.
29. 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, Berlin, 2008. Springer.
30. 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.
31. 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, Berlin, 2006. Springer.
32. 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, Berlin, 2013. Springer.
33. Swen Jacobs and Uwe Waldmann. Comparing instance generation methods for automated reasoning. *Journal of Automated Reasoning*, 38:57–78, 2007.
34. Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Proceedings of the Conference on Computational Problems in Abstract Algebras*, pages 263–298. Pergamon Press, Oxford, 1970.
35. 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, Berlin, 2009. Springer.
36. 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, Berlin, 2013.
37. 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, Berlin, 2010. Springer.
38. 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, Berlin, 2013. Springer.
39. Shie-Jue Lee and David A. Plaisted. Eliminating duplication with the hyperlinking strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.
40. 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 Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 381–385, Berlin, 2001. Springer.
41. 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, Berlin, 2001. Springer.

42. 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, Berlin, 2002. Springer.
43. Michel Ludwig and Uwe Waldmann. An extension of the Knuth-Bendix ordering with LPO-like properties. In Nachum Dershowitz and Andrei Voronkov, editors, *Proceedings of the 14th International Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 4790 of *Lecture Notes in Artificial Intelligence*, pages 348–362, Berlin, 2007. Springer.
44. Michael J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. Technical report, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA, 1988.
45. Michael J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proceedings of the 3rd Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 348–457. IEEE Computer Society Press, 1988.
46. Sharad Malik and Lintao Zhang. Boolean satisfiability: from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76–82, 2009.
47. Rainer Manthey and François Bry. SATCHMO: a theorem prover implemented in Prolog. In Ewing Lusk and Ross Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction (CADE)*, volume 310 of *Lecture Notes in Computer Science*, pages 415–434, Berlin, 1988. Springer.
48. João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marjin Heule, Hans Van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 4, pages 131–153. IOS Press, 2009.
49. João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.
50. William W. McCune. OTTER 3.0 reference manual and guide. Technical Report 94/6, MCS Division, Argonne National Laboratory, 1994.
51. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 39th Design Automation Conference (DAC)*, pages 530–535, 2001.
52. Ruzica Piskac, Leonardo de Moura, and Nikolaj Bjørner. Deciding effectively propositional logic using DPLL and substitution sets. *Journal of Automated Reasoning*, 44(4):401–424, 2010.
53. David A. Plaisted. Mechanical theorem proving. In Ranan B. Banerji, editor, *Formal Techniques in Artificial Intelligence*, pages 269–320. Elsevier, Amsterdam, 1990.
54. 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, Berlin, 2013.
55. David A. Plaisted and Yunshan Zhu. Ordered semantic hyper linking. *Journal of Automated Reasoning*, 25:167–217, 2000.
56. Giles Reger, Martin Suda, and Andrei Voronkov. Playing with AVATAR. In Amy P. Felty and Aart Middeldorp, editors, *Proceedings of the 25th International Conference on Automated Deduction (CADE)*, volume 9195 of *Lecture Notes in Artificial Intelligence*, pages 399–415, Berlin, 2015. Springer.
57. Alexandre Riazanov. *Implementing an efficient theorem prover*. PhD thesis, Department of Computer Science, The University of Manchester, July 2003.
58. J. Alan Robinson. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1:227–234, 1965.
59. 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, Berlin, 2013. Springer.
60. James R. Slagle. Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 14(4):687–697, 1967.
61. Christoph Weidenbach. Combining superposition, sorts and splitting. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, pages 1965–2012. Elsevier, Amsterdam, 2001.

62. 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, Berlin, 2009. Springer.
63. 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.