Constraint Manipulation in SGGS

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

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

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

Abstract

SGGS (Semantically-Guided Goal-Sensitive theorem proving) is a clausal theorem-proving method, with a seemingly rare combination of properties: it is first order, DPLL-style model based, semantically guided, goal sensitive, and proof confluent. SGGS works with *constrained clauses*, and uses a *sequence* of constrained clauses to represent a tentative model of the given set of clauses. A basic building block in SGGS inferences is *splitting*, which partitions a clause into clauses that have the same set of ground instances. Splitting introduces constraints and their manipulation, which is the subject of this paper. Specifically, splitting a clause with respect to another clause requires to compute their *difference*, which captures the ground instances of one that are not ground instances of the other. We give a set of inference rules to compute clause difference, and reduce SGGS constraints to *standard form*, and we prove that it is guaranteed to terminate, provided the standardization rules are applied within the clause difference computation.

Introduction

The SGGS theorem-proving method combines instance generation, resolution, and constraint solving in a *model-based* framework. It works with a set S of first-order clauses to be refuted and an *initial interpretation* I for *semantic guidance*. The features of SGGS can be seen as an attempt to build a model of S, distinct from I. The search for a model of S is done by constructing an SGGS derivation, which is a series $\Gamma_0 \vdash \Gamma_1 \vdash \Gamma_2 \vdash \ldots$ of objects Γ, called *SGGS clause sequences*. After Γ_0, which is empty, each Γ_i is obtained from the previous one by an SGGS inference rule.

An SGGS derivation terminates, if either a refutation is found, or no more inference rules can be applied. SGGS is *refutationally complete*: if S is unsatisfiable, there exist SGGS derivations from S that terminate with the generation of the empty clause. If S is satisfiable, the derivation may be infinite, and if so will in the limit represent a model of S. At each step the new clause sequence replaces the old one, so that only one clause sequence exists at any time, and SGGS is *proof confluent*: performing an inference will never prevent it from finding a refutation, so that *there is no need for backtracking*.

A key property of SGGS is that an SGGS clause sequence represents a candidate partial model. While in propositional logic, a model is represented by a sequence of literals (e.g., as in DPLL), in SGGS a first-order model is represented by a sequence of *constrained clauses*, each of which has a *selected literal*. The model $I[\Gamma]$ represented by a sequence Γ is given by the initial interpretation I modified to satisfy ground instances of selected literals. Informally, the literal L selected in the n-th clause C in Γ contributes to $I[\Gamma]$ its ground instances $L\sigma$ that are needed ($C\sigma$ is not already true in the model induced by the first $n-1$ clauses of Γ) and consistent ($\neg L\sigma$ is not true in the model induced by the first $n-1$ clauses of Γ). Formally, $I[\Gamma]$ is defined inductively on the length of the sequence [2, 1].

The inference rules of SGGS implement a search for a model thus represented. The main rule is *extension*, which adds to the current clause sequence an instance of a clause in S: the

objective is to find a model of all instances of all clauses in S, and if some are not satisfied, they must be added.

It may happen that selected literals have ground instances in common. If the literals have opposite sign, this would make the model *inconsistent*: SGGS features a restricted form of resolution, called *SGGS-resolution*, to remove such contradictions. SGGS-resolution represents an implicit sort of backtracking over the set of possible models of S. The resolvent is a *lemma*, that constrains the model, because the model must satisfy it, and intuitively captures a portion of the search space of models that has been explored. If resolution generates the empty clause, no model can be found.

If selected literals have ground instances in common, and have the same sign, there is *duplication*. SGGS features *splitting rules* that partition a clause with respect to another clause. The clause that gets partitioned, or split, is replaced by other clauses, that have its same set of ground instances, in such a way that the duplicated literals are isolated and can be removed.

The splitting rules of SGGS are the motivation for this paper. The splitting of a clause with respect to another clause can be computed by computing unification of selected literals, and the *difference* between two clauses. Intuitively, the splitting of C with respect to D replaces C by a set of clauses one of which captures the (constrained) ground instances of C that are also (constrained) ground instances of D, while the others capture the (constrained) ground instances of C that are not (constrained) ground instances of D. The latter form the difference between C and D, which is what matters in practice, since we want to remove the duplication.

In this paper, we illustrate the ingredients of SGGS that are relevant to constraint solving: SGGS constraints, constrained clauses, and the concepts of splitting of clauses and difference between clauses. We give a system of rules for constraint manipulation to compute clause differences, whence splittings, and reduce SGGS constraints to standard form. Then we discuss *termination*: while unrestricted applications of the standardization rules may not terminate, the computation of clause difference, and restoration of standard form during the computation of clause difference, are proved to terminate.

For the interested reader, a technical presentation of SGGS, including inference system, fairness, and proofs of refutational completeness and goal-sensitivity, is available in [2]. A non-technical exposition is offered in [3]. The representation of models by SGGS clause sequences is studied in its own right in [1].

Constrained Clauses and Splitting

We assume standard concepts and notations in clausal theorem proving. In addition, $\equiv$ is syntactic identity; $top(t)$ is the top symbol of term t; $at(L)$ is the atom of literal L; $at(T) = \{at(L) : L \in T\}$ for T a set of literals; $vars(C)$ is the set of variables in clause C, and the same notation applies to terms; clauses are *variants*, if made identical by a variable renaming, *similar*, if made identical by a substitution that replaces variables by variables, but may replace distinct variables by the same.

SGGS Constraints

In SGGS, an *atomic constraint* is either empty, denoted by *true*, or an expression of the form $x \equiv y$ or $top(t) = f$, where x and y are variables, f is a function symbol, and t is a term. Then, a *constraint* is either an atomic constraint, or the negation, conjunction, or disjunction of constraints.

SGGS constraints assume *Herbrand interpretations*: let $\models$ mean truth in all Herbrand interpretations; then, $\models t \equiv u$ for ground terms t and u if t and u are the same element of the

Herbrand universe; and $\models top(t) = f$ if the top symbol of ground term t is f. Thus, if $A\vartheta$ is a ground instance of a constraint A, either $\models A\vartheta$ or $\models \neg A\vartheta$.

SGGS constraints are a variant of *Herbrand constraints*: they are *Herbrand constraints* with the addition of atomic constraints of the form $top(t) = f$, which allow us to avoid existential quantifiers in the constraints, since $top(t) = f$ replaces $\exists x_1 \ldots \exists x_n.\ t \equiv f(x_1, \ldots, x_n)$.

An SGGS constraint is in *standard form*, if it is a conjunction of distinct atomic constraints of the form $x \not\equiv y$ and $top(x) \neq f$, where x and y are variables. A constraint $top(x) \neq f$ says that x cannot be replaced by a term whose top function symbol is f, while a constraint $x \not\equiv y$ specifies that x and y may not be replaced by identical terms.

A *constrained clause* is a formula $A \rhd C$, where A is a constraint and C is a clause; a variable that appears in A but not in C is implicitly existentially quantified. A constrained clause $A \rhd C$ may have a *selected* literal L, written $A \rhd C[L]$. $A \rhd L$ is called *constrained literal*. By convention, if L is selected in C, and $C' \equiv C\vartheta$, then $L' \equiv L\vartheta$ is selected in C'.

The *constrained ground instances* (cgi) of $A \rhd C$ are the ground instances of C that satisfy the constraints: $Gr(A \rhd C) = \{C\vartheta :\ \models A\vartheta,\ C\vartheta\ ground\}$, where $\models$ means truth in all Herbrand interpretations. Similarly, $Gr(A \rhd L) = \{L\vartheta :\ \models A\vartheta,\ L\vartheta\ ground\}$. For example, $P(a, b) \in Gr(x \not\equiv y \rhd P(x, y))$, but $P(b, b) \notin Gr(x \not\equiv y \rhd P(x, y))$. A constrained clause (literal) represents its constrained ground instances.

Partition, Splitting, and Difference

Since SGGS uses constrained literals and clauses to exhibit a partial model, it needs to know when constrained literals have instances in common: $A \rhd L$ and $B \rhd M$ *intersect* if $at(Gr(A \rhd L)) \cap at(Gr(B \rhd M)) \neq \emptyset$, and are *disjoint*, otherwise. Intersection does not require the literals to have the same sign, because it is defined based on atoms.

If $A \rhd L$ and $B \rhd M$ do not share variables, they intersect if and only if $at(L)$ and $at(M)$ unify and $(A \wedge B)\sigma$ is satisfiable, where σ is the mgu (most general unifier) of $at(L)$ and $at(M)$. The intersection is given by $at(Gr(A \rhd L)) \cap at(Gr(B \rhd M)) = at(Gr((A \wedge B)\sigma \rhd M\sigma)) = Gr((A \wedge B)\sigma \rhd at(M)\sigma)$.

A *partition* of $A \rhd C[L]$, where A is satisfiable, is a set $\{A_i \rhd C_i[L_i]\}_{i=1}^n$ such that $Gr(A \rhd C) = \bigcup_{i=1}^n \{Gr(A_i \rhd C_i[L_i])\}$, the $A_i \rhd L_i$'s are pairwise disjoint, the A_i's are satisfiable, and the L_i's are chosen consistently with L.

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

$$\{true \rhd [P(f(z), y)] \vee Q(f(z), y),\ top(x) \neq f \rhd [P(x, y)] \vee Q(x, y)\}$$

is a partition of $true \rhd [P(x, y)] \vee Q(x, y)$. On the other hand,

$$\{true \rhd P(f(z), y) \vee [Q(f(z), y)],\ top(x) \neq f \rhd P(x, y) \vee [Q(x, y)]\}$$

is not a partition of $true \rhd [P(x, y)] \vee Q(x, y)$, because selected literals are not chosen consistently.

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

1. $\exists j$, $1 \leq j \leq n$, 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.

The *difference* $C - D$ is $split(C, D)$ with C_j removed. Clause C_j is the *representative* of D in $split(C, D)$: $at(Gr(A_j \rhd L_j))$ is the intersection of $A \rhd L$ and $B \rhd M$, while $C - D$ captures the cgi's of L that are not cgi's of M. We write $Gr(C - D)$ for $\bigcup_{i=1, i\neq j}^{n} Gr(C_i)$.
For example, a splitting of $true \rhd P(x, y)$ by $true \rhd P(f(w), g(z))$ is

$$\{true \rhd P(f(w), g(z)),\ top(x) \neq f \rhd P(x, y),\ top(y) \neq g \rhd P(f(x), y)\}$$

and their difference is $\{top(x) \neq f \rhd P(x, y),\ top(y) \neq g \rhd P(f(x), y)\}$. 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 $true \rhd P(x, y)$ by $true \rhd P(f(w), g(z))$, 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 cgi of both. In the correct splitting, $P(a, b)$ is a cgi of $top(x) \neq f \rhd P(x, y)$, not of $top(y) \neq g \rhd P(f(x), y)$.

As this example shows, computing $split(C, D)$ and $C - D$ introduces constraints, including non-standard ones, even when C and D have empty constraints to begin with. This is precisely why SGGS works with constrained clauses.

If $at(L)$ and $at(M)$ do not unify, $Gr(C - D) = Gr(C)$; if they unify with mgu σ, then $split(C, D) = (C - D) \cup \{A\sigma \wedge B\sigma \rhd C[L]\sigma\}$, and $(C - D) = (C - (A\sigma \wedge B\sigma \rhd C[L]\sigma))$. Thus, if we have a way to compute $C - D$, we also have a way to compute $split(C, D)$, and we can restrict ourselves to compute $C - D$ under the assumption that D is an instance of C.

Rules to Compute Clause Difference and Standardize Constraints

The following rules are *sound*, as premise and conclusion represent the same set of cgi's. If a conclusion has the form $A_1 \rhd C_1, \ldots, A_n \rhd C_n$, it is a disjunction, and represents $\bigcup_{i=1}^{n} Gr(A_i \rhd C_i)$. We begin with rules to compute $C - D$ when $D \equiv C\sigma$.

Rules for Clause Difference and Disjunctive Normal Form

If $\{x \leftarrow f(x_1, \ldots, x_n)\} \subseteq \sigma$ for some $x \in vars(C)$ and new variables x_i, $1 \leq i \leq n$, 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 $\{x \leftarrow y\} \subseteq \sigma$ for distinct variables $x, y \in vars(C)$, 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:

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

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. The next rules restore disjunctive normal form (DNF). The *Equiv rule* replaces a constraint by its DNF:

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

where $dnf(A)$ is the disjunctive normal form of A; and the *Div rule* subdivides disjunction:

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

Rules for Reduction to Standard Form

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 standard form $x \not\equiv y$.
The *ElimId1 rule* eliminates a constraint between variable and term: if $x \notin vars(s)$, then:

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

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

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

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

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

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

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:

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

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:

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

As an example, consider computing $split(C, D)$, where $A \rhd C[L]$ is $true \rhd P(x, f(x))$ and $B \rhd D[M]$ is $x \not\equiv y \rhd P(x, y)$. After renaming variables in the second clause, so that $B \rhd D[M]$ becomes $x' \not\equiv y \rhd P(x', y)$, the unification of $at(L) = P(x, f(x))$ and $at(M) = P(x', y)$, yields mgu $\sigma = \{x' \leftarrow x, y \leftarrow f(x)\}$, so that $A\sigma \wedge B\sigma \rhd C[L]\sigma$ is $x \not\equiv f(x) \rhd P(x, f(x))$. The ElimId1 rule reduces this clause to $true \rhd P(x, f(x))$, which is the same as $A \rhd C[L]$. Thus, $C - D = C - C$, or the difference is empty, because indeed $Gr(A \rhd C[L]) \subseteq Gr(B \rhd D[M])$. Accordingly, $split(C, D)$ is $A \rhd C[L]$ itself, or the splitting operation leaves the clause unchanged, because we tried to split a clause by a more general one.

The *rules for top symbol* eliminate all top symbol constraints, except those in standard form $top(x) \neq f$.
The *ElimTop1 rule* detects a conflict in a positive constraint: if $f \neq g$, $n \geq 0$, then:

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

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:

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

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 for all i, $1 \leq i \leq n$, x_i is a new variable.
The combined effect of all rules is to standardize all constraints.

Termination

The application of the identity rules may not terminate in general. For example, consider a clause $(x \not\equiv f(y) \wedge y \not\equiv f(x) \rhd P(x, y))$: 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.

Nonetheless, SGGS does not need that every series of applications of these rules terminate. It suffices that the computation of clause difference terminates:

Theorem 1. *Given $A \rhd C$ and $B \rhd D$, such that $D \equiv C\sigma$, 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 $f(x_1, \ldots, x_n) \not\equiv y$, enabling 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.

□

Discussion

We presented a set of inference rules to compute the difference between two constrained clauses, and to reduce to standard form SGGS constraints. We showed by a counter-example that it is not the case that any application of the inference rules for standardization is guaranteed to terminate. Then we proved that computation of clause difference is guaranteed to terminate, and that standardization in the context of computing clause differences is also guaranteed to terminate.

SGGS is a new reasoning method that uses sequences of constrained clauses to represent candidate partial models, during the search for a refutation, or a model, of a set of first-order clauses. When clauses in the sequence contribute to the candidate partial model sets of ground literals with non-empty intersection, there is a duplication. SGGS removes this duplication by inferences that split a clause with respect to another. Computing this splitting requires to compute unification of literals and differences of clauses, whence the interest for the difference operation.

SGGS constraints are a variant of Herbrand constraints (e.g., [6, 7, 5, 4]): they feature atomic constraints in the form $top(t) = f$, which allow one to avoid existential quantifiers in

constraints. If $top(t) = f$ is replaced with $\exists x_1 \ldots \exists x_n.\ t \equiv f(x_1, \ldots, x_n)$, SGGS constraints fit in the first-order logic of equations between trees.

Inference systems to decide the truth in the Herbrand universe of first-order formulæ with equality as the only predicate symbol were given independently in [6, 7] and [5]. Our inference system and termination result are tailored for the SGGS reasoning method; they capture what is needed precisely by SGGS, and therefore they are relevant to its understanding and implementation. More study may clarify a more precise relationship between our work in this paper and that in [6, 7] and [5]. Another possible topic for future investigation is the complexity of these procedures. The research in [6, 7] and [5] was motivated primarily by logic programming with constraints. It is interesting that those results may turn out to be useful for a theorem-proving method after several years.

References

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

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

[3] Maria Paola Bonacina and David A. Plaisted. SGGS theorem proving: an exposition. In Leonardo De Moura Boris Konev and Stephan Schulz, editors, *Notes of the Fourth Workshop on Practical Aspects in Automated Reasoning (PAAR), Seventh International Joint Conference on Automated Reasoning (IJCAR) and Sixth Federated Logic Conference (FLoC)*, EasyChair Proceedings in Computing (EPiC), pages 1–14, July 2014.

[4] 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, 1991.

[5] Hubert Comon and Pierre Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7:371–425, 1989.

[6] Michael J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. Technical report, IBM, Thomas J. Watson Research Center, Yorktown Heights, New York, USA, 1988.

[7] Michael J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proceedings of the Third Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 348–457. IEEE Computer Society Press, 1988.