DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD POLITÉCNICA DE VALENCIA

P.O. Box: 22012 E-46071 Valencia (SPAIN)

Departamento de Sistemas
Informáticos y Computación

Informe Técnico / Technical Report

Ref. No.:	DSIC-II/06/08	**Pages:10**
Title:	A Modular Equational Generalization Algorithm	
Author(s):	M. Alpuente, S. Escobar, J. Mesesguer and P. Ojeda	
Date:	09/05/2008	
Keywords:	least general generalization, rule-based languages, equational reasoning	

V^{w}o B^{w}o
Leader of research Group Author(s)

A Modular Equational Generalization Algorithm*

María Alpuente[1], Santiago Escobar[1], José Meseguer[2], and Pedro Ojeda[1]

[1] Universidad Politécnica de Valencia, Spain.
{alpuente,sescobar,pojeda}@dsic.upv.es
[2] University of Illinois at Urbana–Champaign, USA. meseguer@cs.uiuc.edu

Abstract. An important component for ensuring termination of many program manipulation techniques is the computation of least general generalizations. In this paper, we present a modular equational generalization algorithm, where function symbols can have any combination of associativity, commutativity, and identity axioms (including the empty set). This is suitable for dealing with functions that obey algebraic laws, and are typically mechanized by means of equational atributes in rule-based languages such as ASF+SDF, Elan, OBJ, Cafe-OBJ, and Maude. The algorithm computes a complete set of least general generalizations modulo the given equational axioms, and is specified by a set of inference rules that we prove correct. This work opens up new applications of generalization to rule-based languages, theorem provers and program manipulation tools such as program analizers, where function symbols obey algebraic axioms. A Web tool which implements the algorithm has been developed which is publicly available.

1 Introduction

The problem of ensuring termination of program manipulation techniques arises in many areas of computer science, including automatic program analysis, synthesis, verification, specialisation, and transformation. An important component for ensuring termination of these techniques is a generalization algorithm (also called anti–unification) that, for a pair of input expressions, returns its least general generalization (lgg), i.e., a generalization that is more specific than any other such generalization. Whereas unification produces most general unifiers that, when applied to two expressions, make them equivalent to the most general common instance of the inputs [18], generalization abstracts the inputs by computing their most specific generalization. As in unification, where the most general unifier (mgu) is of interest, in the sequel we are interested in the least general generalization (lgg) or, as we shall see for the equational case treated in this paper, in a minimal and complete set of lggs, which is the dual analogue of a minimal and complete set of unifiers for equational unification problems.

For instance, in the partial evaluation (PE) of logic programs [14], the general idea is to construct a set of finite (possibly partial) deduction trees for a set of initial

* This work has been partially supported by the EU (FEDER) and the Spanish MEC TIN2007-68093-C02-02 project, UPV PAID-06-07 project, and Generalitat Valenciana GV06/285 and BFPI/2007/076 grants.

calls, and then extract from those trees a new program P that allows any instance of the calls to be executed. To ensure that the partially evaluated program *covers* all these calls, most PE procedures recursively specialize some calls that are dynamically produced during this process. This requires some kind of generalization in order to enforce the termination of the process: if a call occurring in P that is not sufficiently covered by the program *embeds* an already evaluated call, both calls are generalized by computing their lgg. In the literature on partial evaluation, least general generalization is also knows as *most specific generalization* (msg) and *least common anti–instance* (lcai) [21].

The computation of lggs is also central to most program synthesis and learning algorithms such as those developed in the area of inductive logic programming [22], and also to conjecture lemmas in inductive theorem provers such as Nqthm [7] and its ACL2 extension [17]. Least general generalization was originally introduced by Plotkin in [24], see also [27]. Actually, Plotkin's work originated from the consideration in [26] that, since unification is useful in automatic deduction by the resolution method, its dual might prove helpful for induction.

Quite often, however, all the above applications of generalization may have to be carried out in contexts in which the function symbols satisfy certain *equational axioms*. For example, in rule-based languages such as ASF+SDF [5], Elan [6], OBJ [15], CafeOBJ [11], and Maude [8] some function symbols may be declared to obey given algebraic laws (the so-called *equational attributes* of OBJ, CafeOBJ and Maude), whose effect is to compute with equivalence classes modulo such axioms while avoiding the risk of non–termination. Similarly, theorem provers, both general first-order logic ones and inductive theorem provers, routinely support commonly occurring equational theories for some function symbols such as associativity-commutativity. In yet another area, [12,13] describes rule-based applications to security protocol verification, where symbolic reachability analyses modulo algebraic properties allow one to reason about security in the face of attempted attacks on low-level algebraic properties of the functions used in the protocol (e.g. commutative encryption). A survey of algebraic properties used in cryptographic protocols can be found in [10]. Surprisingly, unlike the dual case of equational unification, which has been thoroughly investigated (see, e.g., the surveys [28,4]), to the best of our knowledge there seems to be no treatment of generalization modulo an equational theory E. This paper makes a novel contribution in this area by developing a modular family of E-generalization algorithms where the theory E is parametric: any binary function symbol can have any combination of associativity, commutativity, and identity axioms (including the empty set of such axioms).

To better motivate our work, let us first recall the standard generalization problem. Let t_1 and t_2 be two terms. We want to find a term s that generalizes both t_1 and t_2. In other words, both t_1 and t_2 must be substitution instances of s. Such a term is, in general, not unique. For example, let t_1 be the term $f(f(a,a),b)$ and let t_2 be $f(f(b,b),a)$. Then $s = x$ trivially generalizes the two terms, with x being a variable. Another possible generalization is $f(x,y)$, with y being also a variable. The term $f(f(x,x),y)$ has the advantage of being the most 'specific' or *least general generalization (lgg)* (modulo variable renaming).

In the case where the function symbols do not satisfy any algebraic axioms, the lgg is always unique. However, when we want to reason *modulo* certain axioms for the different function symbols in our problem, lggs no longer need to be unique. This is analogous to equational unification problems where in general there is no single mgu, but a set of them. Let us, for example, consider that the above function symbol f is associative and commutative. Then the term $f(f(x,x),y)$ is not the only least general generalization of $f(f(a,a),b)$ and $f(f(b,b),a)$, because another incomparable generalization exists, namely, $f(f(x,a),b)$.

Similarly to the case of equational unification [28], things are not so easy as for syntatic generalization, and the dual problem of computing least general E-generalizations is a difficult problem, particularly in managing the algorithmic complexity of the problem. The significance of equational generalization was already pointed out by Pfenning in [23]: "It appears that the intuitiveness of generalizations can be significantly improved if anti–unification takes into account additional equations which come from the object theory under consideration. It is conceivable that there is an interesting theory of equational anti–unification to be discovered". In this work, we do not address the E-generalization problem in its fullest generality. Instead, we study in detail a *modular* algorithm for a *parametetric* family of commonly occurring equational theories, namely, for all theories (Σ, E) such that each binary function symbol $f \in \Sigma$ can have any combination of the following equational axioms: (i) *associativity* (A_f) $f(x, f(y,z)) = f(f(x,y),z)$; (ii) *commutativity* (C_f) $f(x.y) = f(y,x)$, and (iii) *identity* (U_f) for a constant symbol, say, e, i.e., $f(x,e) = x$ and $f(e,x) = x$. In particular, f may not satisfy any such axioms, which when it happens for all binary symbols $f \in \Sigma$ gives us the standard generalization algorithm as a special case.

The main contributions of the paper can be summarized as follows:

- A modular equational generalization algorithm specified as a set of inference rules, where different function symbols satisfying different associativity and/or commutativity and/or identity axioms have different inference rules.
- A proof of correctness for the inference rules.

The algorithm is *modular* in the precise sense that the combination of different equational axioms for different function symbols is automatic and seamless: the inference rules can be applied to generalization problems involving each symbol with no need whatsoever for any changes or adaptations. This is similar to, but much simpler and easier than, modular methods for combining E-unification algorithms (see, e.g., [4]). We illustrate our inference system with several examples.

As already mentioned, our E-generalization algorithm should be of great interest to developers of rule-based languages, theorem provers and equational reasoning programs, as well as program manipulation tools such as program analyzers and partial evaluators, for declarative languages and reasoning systems supporting commonly occurring equational axioms such as associativity, commutativity and identity efficiently in a built–in way. For instance, this includes many theorem provers, and a variety of rule-based languages such as ASF+SDF, OBJ, CafeOBJ, Elan, and Maude.

After some preliminaries in Section 2, we present in Section 3 a syntactic generalization algorithm as a special case to motivate its equational extension. Then in

Section 4, we show how this calculus naturally extends to a new, modular generalization algorithm modulo ACU. We illustrate the use of the inference rules with several examples. Finally, we give a proof of correctness of our inference system. Section 6 concludes.

2 Preliminaries

We follow the classical notation and terminology from [29] for term rewriting and from [19,20] for rewriting logic. We assume an *unsorted signature* Σ with a finite number of function symbols. We assume an enumerable set of variables $\mathcal{X}$. A *fresh* variable is a variable that appears nowhere else. We write $\mathcal{T}(\Sigma, \mathcal{X})$ and $\mathcal{T}(\Sigma)$ for the corresponding term algebras. For a term t, we write $Var(t)$ for the set of all variables in t. The set of positions of a term t is written $Pos(t)$, and the set of non-variable positions $Pos_\Sigma(t)$. The root position of a term is Λ. The subterm of t at position p is $t|_p$ and $t[u]_p$ is the term t where $t|_p$ is replaced by u. By $root(t)$ we denote the symbol occurring at the root position of t.

A *substitution* σ is a mapping from a finite subset of $\mathcal{X}$, written $Dom(\sigma)$, to $\mathcal{T}(\Sigma, \mathcal{X})$. The set of variables introduced by σ is $Ran(\sigma)$. The identity substitution is *id*. Substitutions are homomorphically extended to $\mathcal{T}(\Sigma, \mathcal{X})$. The application of a substitution σ to a term t is denoted by $t\sigma$. The restriction of σ to a set of variables V is $\sigma|_V$. Composition of two substitutions is denoted by juxtaposition, i.e., $\sigma\sigma'$. We call a substitution σ a *renaming* if there is another substitution σ^{-1} such that $\sigma\sigma^{-1}|_{Dom(\sigma)} = id$.

A *rewrite rule* is an oriented pair $l \to r$, where $l \notin \mathcal{X}$, and $Var(r) \subseteq Var(l)$. An *(unconditional) rewrite theory* is a tuple $\mathcal{R} = (\Sigma, R)$ with Σ a signature, and R a set of rewrite rules. The rewriting relation on $\mathcal{T}(\Sigma, \mathcal{X})$, written $t \xrightarrow{p}_R t'$ (or $t \to_R t'$) holds between t and t' iff there exist $p \in Pos_\Sigma(t)$, $l \to r \in R$ and a substitution σ, such that $t|_p = l\sigma$, and $t' = t[r\sigma]_p$.

A *Σ-equation* is an unoriented pair $t = t'$. An *equational theory* (Σ, E) is a set of Σ-equations. An equational theory (Σ, E) is *regular* if for each $t = t' \in E$, we have $Var(t) = Var(t')$. Given Σ and a set E of Σ-equations, equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}(\Sigma, \mathcal{X})$ (see [20]).

3 Syntactic Least General Generalization

Plotkin [24] and Reynolds [27] gave an imperative–style algorithm for generalization, which are both essentially the same. Huet gave a new generalization algorithm [16] formulated as a pair of recursive equations. In this section, we revisit syntactic generalization, giving a novel inference system that will be useful in our subsequent extension of this algorithm to the equational setting given in Section 4.

Let $\leq$ be the standard instantiation quasi–ordering on terms given by the relation of being "more general", i.e. t is more general than s (i.e. s is an instance of t), written $t \leq s$, iff there exists θ such that $t\theta = s$. Most general unification of a (unifiable) set M is the least upper bound (most general instance) of M under $\leq$. Generalization corresponds to the greatest lower bound. Given a non–empty set M of terms, the

term w is a generalization of M if, for all $s \in M$, $w \leq s$. A term w is the least general generalization of M if w is a generalization of M and, for each other generalization u of M, $u \leq w$.

The non-deterministic generalization algorithm λ of Huet [16] (also treated in detail in [18]) contains some implicit (imprecise) assumptions in the algorithm regarding the treatment of variables that are hidden in a global function Φ. In [1], we have provided a *novel* set of inference rules for computing the (syntactic) least generalization of two terms, that avoids implicit, hidden assumptions by using a store of already solved generalization sub-problems. The advantage of using such a store is that, differently from the global repository Φ, our stores are local to the computation traces. This non–globality of the stores is the key for both, the order–sorted version of [1] and the equational generalization algorithm developed in this work that computes a complete and minimal set of least general E-generalizations.

In our reformulation [1], we represent a generalization problem between terms s and t as a *constraint* $s \overset{x}{\triangleq} t$, where x is a fresh variable that stands for the (most general) generalization of s and t. By means of this representation, any generalization w of s and t is given by a substitution θ such that $x\theta = w$.

We compute the least general generalization of s and t, written $lgg(s, t)$, by means of a transition system $(Conf, \rightarrow)$ [25] where $Conf$ is a set of *configurations* and the transition relation $\rightarrow$ is given by a set of inference rules. Besides the *constraint component*, i.e., a set of constraints of the form $t_i \overset{x_i}{\triangleq} t_{i'}$, and the *substitution component*, i.e., the partial substitution computed so far, configurations also include an extra component, called the *store*.

Definition 1. *A configuration, written as $\langle CT \mid S \mid \theta \rangle$, consists of three components:*

- *the* constraint component CT*, i.e., a conjuntion $s_1 \overset{x_1}{\triangleq} t_1 \wedge \ldots \wedge s_n \overset{x_n}{\triangleq} t_n$ that represents the* set of unsolved constraints
- *the* store component S*, that records the* set of already solved constraints*, and*
- *the* substitution component θ*, that consists of bindings for some variables previously met during the computation.*

Starting from the initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle$, configurations are transformed until a terminal configuration $\langle \emptyset \mid S \mid \theta \rangle$ is reached. Then, the lgg of t and t' is given by $x\theta$. As we will see, θ is unique up to renaming.

The transition relation $\rightarrow$ is given by the smallest relation satisfying the rules in Figure 1. In this paper, variables of terms t and s in a generalization problem $t \overset{x}{\triangleq} s$ are considered as constants, and are never instantiated. The meaning of the rules is as follows.

- The rule **Decompose** is the syntactic decomposition generating new constraints to be solved.
- The rule **Recover** checks if a constraint $t \overset{x}{\triangleq} s \in CT$ with $root(t) \not\equiv root(s)$, is already solved, i.e., there is already a constraint $t \overset{y}{\triangleq} s \in S$ for the same *conflict pair* (t, s), with variable y. This is needed when the input terms of the generalization

problem contain the same conflict pair more than once, e.g., the lgg of $f(a,a,a)$ and $f(b,b,a)$ is $f(y,y,a)$.

– The rule **Solve** checks that a constraint $t \overset{x}{\triangleq} s \in CT$ with $root(t) \not\equiv root(s)$, is not already solved. If not already there, the solved constraint $t \overset{x}{\triangleq} s$ is added to the store S.

Note that the inference rules of Figure 1 are non–deterministic (i.e., they depend on the chosen constraint of the set CT). However, they are confluent up to variable renaming (i.e., the chosen transition is irrelevant for computation of terminal configurations). This justifies that the least general generalization of two terms is unique up to variable renaming [18].

Decompose

$$\frac{f \in (\Sigma \cup \mathcal{X})}{\langle f(t_1,\ldots,t_n) \overset{x}{\triangleq} f(t'_1,\ldots,t'_n) \wedge CT \mid S \mid \theta\rangle \rightarrow \langle t_1 \overset{x_1}{\triangleq} t'_1 \wedge \ldots \wedge t_n \overset{x_n}{\triangleq} t'_n \wedge CT \mid S \mid \theta\sigma\rangle}$$

where $\sigma = \{x \mapsto f(x_1,\ldots,x_n)\}$, $x_1,\ldots,x_n$ are fresh variables, and $n \geq 0$

Solve

$$\frac{root(t) \not\equiv root(t') \wedge \nexists y : t \overset{y}{\triangleq} t' \in S}{\langle t \overset{x}{\triangleq} t' \wedge CT \mid S \mid \theta\rangle \rightarrow \langle CT \mid S \wedge t \overset{x}{\triangleq} t' \mid \theta\rangle}$$

Recover

$$\frac{root(t) \not\equiv root(t')}{\langle t \overset{x}{\triangleq} t' \wedge CT \mid S \wedge t \overset{y}{\triangleq} t' \mid \theta\rangle \rightarrow \langle CT \mid S \wedge t \overset{y}{\triangleq} t' \mid \theta\sigma\rangle}$$

where $\sigma = \{x \mapsto y\}$

Fig. 1. Rules for least general generalization

Example 1. Let $t = f(g(a), g(y), a)$ and $s = f(g(b), g(y), b)$ be two terms. We apply the inference rules of Figure 1 and the substitution obtained by the lgg algorithm is $\theta = \{x \mapsto f(g(x_4), g(y), x_4), x_1 \mapsto g(x_4), x_2 \mapsto g(y), x_5 \mapsto y, x_3 \mapsto x_4\}$, where the lgg is $x\theta = f(g(x_4), g(y), x_4)$. Note that variable x_4 is repeated, to ensure the least general generalization. The execution trace is showed in Figure 2.

Termination and confluence (up to variable renaming) of the transition system $(Conf, \rightarrow)$ are straightforward. Soundness and completeness is proved as follows.

Theorem 1. [1] *Given terms t and t' and a fresh variable x, u is the lgg of t and t' if and only if $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ and there is a renaming ρ s.t. $u\rho = x\theta$.*

Let us mention that the equational generalization algorithm of [1] recalled above can also be used to compute (thanks to associativity and commutativity of lgg) the lgg of an arbitrary set of terms by successively computing the lgg of two elements of the set in the obvious way.

$$lgg(f(g(a),g(y),a),f(g(b),g(y),b))$$
$$\downarrow \text{ Initial Configuration}$$
$$\langle f(g(a),g(y),a) \overset{x}{\triangleq} f(g(b),g(y),b) \mid \emptyset \mid id\rangle$$
$$\downarrow \text{ Decompose}$$
$$\langle g(a) \overset{x_1}{\triangleq} g(b) \wedge g(y) \overset{x_2}{\triangleq} g(y) \wedge a \overset{x_3}{\triangleq} b \mid \emptyset \mid \{x \mapsto f(x_1,x_2,x_3)\}\rangle$$
$$\downarrow \text{ Decompose}$$
$$\langle a \overset{x_4}{\triangleq} b \wedge g(y) \overset{x_2}{\triangleq} g(y) \wedge a \overset{x_3}{\triangleq} b \mid \emptyset \mid \{x \mapsto f(g(x_4),x_2,x_3), x_1 \mapsto g(x_4)\}\rangle$$
$$\downarrow \text{ Solve}$$
$$\langle g(y) \overset{x_2}{\triangleq} g(y) \wedge a \overset{x_3}{\triangleq} b \mid a \overset{x_4}{\triangleq} b \mid \{x \mapsto f(g(x_4),x_2,x_3), x_1 \mapsto g(x_4)\}\rangle$$
$$\downarrow \text{ Decompose}$$
$$\langle y \overset{x_5}{\triangleq} y \wedge a \overset{x_3}{\triangleq} b \mid a \overset{x_4}{\triangleq} b \mid \{x \mapsto f(g(x_4),g(x_5),x_3), x_1 \mapsto g(x_4), x_2 \mapsto g(x_5)\}\rangle$$
$$\downarrow \text{ Decompose}$$
$$\langle a \overset{x_3}{\triangleq} b \mid a \overset{x_4}{\triangleq} b \mid \{x \mapsto f(g(x_4),g(y),x_3), x_1 \mapsto g(x_4), x_2 \mapsto g(y), x_5 \mapsto y\}\rangle$$
$$\downarrow \text{ Recover}$$
$$\langle \emptyset \mid a \overset{x_4}{\triangleq} b \mid \{x \mapsto f(g(x_4),g(y),x_4), x_1 \mapsto g(x_4), x_2 \mapsto g(y), x_5 \mapsto y, x_3 \mapsto x_4\}\rangle$$

Fig. 2. Computation trace for (syntactic) generalization of terms $f(g(a),g(y),a)$ and $f(g(b),g(y),b)$

4 Least General Generalizations modulo E

When we have an equational theory E, the notion of least general generalization has to be broadened, because, there may exist E-generalizable terms that do not have a unique lgg. Similarly to the dual case of E-unification, we have to talk about a *set* of lggs.

Example 2. Consider terms $t = f(a,a,b)$ and $s = f(b,b,a)$ where f is associative and commutative, and a and b are constants. Terms $u = f(x,x,y)$ and $u' = f(x,a,b)$ are generalizations of t and s but they are not comparable, i.e., no one is an instance of the other modulo the AC axioms of f.

Given a finite equational theory E, and given two terms t and s, we can always recursively enumerate the set which is by construction a complete set of generalizations of t and s. For this, we only need to recursively enumerate all pairs of terms (u,u') with $t =_E u$ and $s =_E u'$. Of course, this set $gen_E(t,s)$ may easily be infinite. However, if the theory E has the additional property that each E-equivalence class is *finite* and can be effectively generated, then the above process becomes a terminating *algorithm*, generating a finite complete set of generalizations of t and s.

In any case, for any finite theory E, we can always mathematically characterize a *minimal complete set* of E-generalizations, namely as a set $lgg_E(t,s)$ defined as follows.

Definition 2. *Let t and s be terms and let E be an equational theory. A complete set of generalizations of t and s modulo E, denoted by $gen_E(t,s)$, is defined as follows:*

$$gen_E(t,s) = \{v \mid \exists u,u' \text{ s.t. } t =_E u \wedge s =_E u' \wedge lgg(u,u') = v\}$$

The set of least general generalizations of t and s modulo E is defined as follows:

$$lgg_E(t,s) = minimal_{<_E}(gen_E(t,s))$$

where $minimal_{<_E}(S) = \{s \in S \mid \nexists s' \in S : s' <_E s\}$. *Lggs are equivalent modulo renaming and, therefore, we remove from* $lgg_E(t, t')$ *renamed versions of terms.*

The following result is immediate.

Theorem 2. *Given terms* t *and* s *in an equational theory* E, $lgg_E(t, s)$ *is a minimal, correct, and complete set of lggs modulo* E *of* t *and* s *(up to renaming).*

However, note that in general the relation $t <_E t'$ is *undecidable*, so that the above set, although definable at the mathematical level, cannot be effectively computed. Nevertheless, when: (i) each E-equivalence class is *finite* and can be effectively generated; and (ii) there is an E-matching algorithm, then we also have an effective algorithm for computing $lgg_E(t, s)$, since the relation $t <_E t'$ is precisely the E-matching relation.

In summary, therefore, when E is finite and satisfies conditions (i) and (ii), the above definitions give us an effective, although horribly inefficient, procedure to compute a finite, minimal, and complete set of least general generalizations $lgg_E(t, s)$. This naive algorithm could be used when E consists of associativity and/or commutativity axioms for some functions symbols, because such theories (a special case of our proposed parametric family of theories) all satisfy conditions (i)–(ii). However, as soon as we add identiy axioms, E-equivalence classes become infinite, so that the above approach no longer gives us an lgg algorithm modulo E.

In the following sections, we do provide a modular, minimal, terminating, sound, and complete algorithm for equational theories containing different axioms such as associativity, commutativity, and identity (and their combinations). The set $lgg_E(t, s)$ of least general E-generalizations is computed in two phases: (i) first a complete set of E-generalizations is computed by the inference rules given below, and then (ii) they are filtered to obtain $lgg_E(t, s)$ by using the fact that for all theories E in the parametric family of theories we consider in this paper, there is a matching algorithm modulo E. We consider that a given function symbol f in the signature Σ obeys a subset of axioms $ax(f) \subseteq \{A_f, C_f, U_f\}$. In particular, f may not satisfy any such axioms, $ax(f) = \emptyset$.

Let us provide our inference rules for equational generalization in a stepwise manner. First, $ax(f) = \emptyset$, then, $ax(f) = \{C_f\}$, then, $ax(f) = \{A_f\}$, then, $ax(f) = \{A_f, C_f\}$, and finally, then, $\{U_f\} \in ax(f)$. Technically, variables of the original terms are handled in our rules as constants, thus without any attribute, i.e., for any variable $x \in X$, we consider $ax(x) = \emptyset$.

4.1 Basic rules for least general E–generalization

Let us start with a set of basic rules that are the equational version of the syntactic generalization rules of Section 3. The rule $Decompose_E$ applies to function symbols obeying no axioms, $ax(f) = \emptyset$. Specific rules for decomposing constraints involving terms that are rooted by symbols obeying equational axioms, such as ACU and their combinations, are given below.

Concerning the rules $Solve_E$ and $Recover_E$, the main difference w.r.t. the corresponding syntactic generalization rules given in Section 3 is in the fact that the

checks to the store consider the constraints modulo E: in the rules below, we write $t \overset{y}{\triangleq} t' \in^E S$ to express that there exists $s \overset{y}{\triangleq} s' \in S$ such that $t =_E s$ and $t' =_E s'$.

Finally, regarding the rule $Solve_E$, note that this rule cannot be applied to any constraint $t \overset{x}{\triangleq} s$ such that either t or s are rooted by a function symbol f with $U_f \in ax(f)$. For function symbols with an identity element, a specially–tailored rule $Expand_U$ is given in Section 4.5 that gives us the opportunity to solve a constraint (conflict pair) $f(t_1, t_2) \overset{x}{\triangleq} s$, such that $root(s) \not\equiv f$, with a generalization $f(y, z)$ more specific than x, by first introducing the constraint $f(t_1, t_2) \overset{x}{\triangleq} f(s, e)$.

Decompose$_E$

$$\frac{f \in (\Sigma \cup \mathcal{X}) \wedge ax(f) = \emptyset}{\langle f(t_1, \ldots, t_n) \overset{x}{\triangleq} f(t'_1, \ldots, t'_n) \wedge CT \mid S \mid \theta \rangle \Rightarrow \langle t_1 \overset{x_1}{\triangleq} t'_1 \wedge \ldots \wedge t_n \overset{x_n}{\triangleq} t'_n \wedge CT \mid S \mid \theta\sigma \rangle}$$

where $\sigma = \{x \mapsto f(x_1, \ldots, x_n)\}$, $x_1, \ldots, x_n$ are fresh variables, and $n \geq 0$

Solve$_E$

$$\frac{f = root(t) \wedge g = root(t') \wedge f \not\equiv g \wedge U_f \notin ax(f) \wedge U_g \notin ax(g) \wedge \nexists y : t \overset{y}{\triangleq} t' \in^E S}{\langle t \overset{x}{\triangleq} t' \wedge CT \mid S \mid \theta \rangle \Rightarrow \langle CT \mid S \wedge t \overset{x}{\triangleq} t' \mid \theta \rangle}$$

Recover$_E$

$$\frac{root(t) \not\equiv root(t') \wedge \exists y : t \overset{y}{\triangleq} t' \in^E S}{\langle t \overset{x}{\triangleq} t' \wedge CT \mid S \mid \theta \rangle \Rightarrow \langle CT \mid S \mid \theta\sigma \rangle}$$

where $\sigma = \{x \mapsto y\}$

Fig. 3. Basic rules for least general E–generalization

4.2 Least general generalization modulo C

In this section we extend the basic set of equational generalization rules by adding a specific inference rule $Decompose_C$, given in Figure 4, for dealing with commutativity function symbols. This inference rule replaces the syntactic decomposition inference rule for the case of a binary commutative symbol f, i.e., the four possible rearrangements of the terms $f(t_1, t_2)$ and $f(s_1, s_2)$ are considered. Just notice that this rule is (don't know) non-deterministic, hence all four combinations must be explored.

Example 3. Let $t = f(a, b)$ and $s = f(b, a)$ be two terms where f is commutative, i.e., $C_f \in ax(f)$. By applying the rules $Solve_E$, $Recover_E$, and $Decompose_C$ above, we end in a terminal configuration $\langle \emptyset \mid S \mid \theta \rangle$, where $\theta = \{x \mapsto f(b, a), x_3 \mapsto b, x_4 \mapsto a\}$, thus we conclude that the lgg modulo C of t and s is $x\theta = f(b, a)$. The computation trace is shown in Figure 5.

Decompose$_C$

$$\frac{C_f \in ax(f) \wedge A_f \notin ax(f) \wedge i \in \{1,2\}}{\langle f(t_1,t_2) \stackrel{x}{=} f(s_1,s_2) \wedge CT \mid S \mid \theta\rangle \Rightarrow \langle t_1 \stackrel{x_1}{=} s_i \wedge t_2 \stackrel{x_2}{=} s_{(i \bmod 2)+1} \wedge CT \mid S \mid \theta\sigma\rangle}$$

where $\sigma = \{x \mapsto f(x_1,x_2)\}$, and x_1, x_2 are fresh variables

Fig. 4. Decomposition rule for a commutative function symbol f

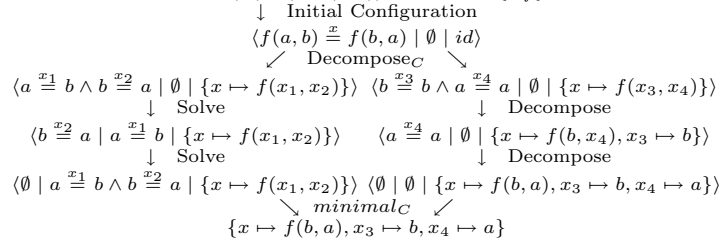

Fig. 5. Computation trace for C–generalization of terms $f(a,b)$ and $f(b,a)$

4.3 Least general generalization modulo *A*

In this section we provide a specific inference rule *Decompose*$_A$ for handling function symbols obeying the associativity axiom (but not the commutativity one). A specific set of rules for dealing with AC function symbols is given in the next subsection.

The *Decompose*$_A$ rule is given in Figure 6. Note that we use flattened versions of the terms which use poly-variadic versions of the associative symbols, i.e., being f an associative symbol,

$$flat(f(t_1,\ldots,f(s_1,\ldots,s_k),\ldots,t_n)) = flat(f(t_1,\ldots,s_1,\ldots,s_k,\ldots,t_n))$$

and, otherwise, $flat(f(t_1,\ldots,t_n)) = f(flat(t_1),\ldots,flat(t_n))$. Given an associative symbol f and a term $f(t_1,\ldots,t_n)$ we call *alien f-terms* (or simply *alien terms*) to those terms among $t_1,\ldots,t_n$ that are not rooted by f. In the following, being f an associative poly-varyadic symbol, $f(t)$ represents the term t itself, since symbol f needs at least two arguments. This inference rule replaces the syntactic decomposition inference rule for the case of an associative function symbol f, where all *prefixes* of $t_1,\ldots,t_n$ and $s_1,\ldots,s_m$ are considered. Just notice that this rule is (don't know) non-deterministic, hence all possibilities must be explored.

Note that this is better than generating all terms in the corresponding equivalence class, as explained in Section 4, since we will eagerly stop in a constraint $t \stackrel{x}{=} f(t_1,\ldots,t_n)$ if $root(t) \not\equiv f$ without considering all the combinations of $f(t_1,\ldots,t_n)$.

We give the rule *Decompose*$_A$ for the case when, in the generalization problem $s \stackrel{x}{\triangleq} t$, the number of *alien terms* in s is greater than (or equal to) the number of alien terms in t. For the other way round, that is, the number of *alien terms* in s is less than (or equal to) the number of alien terms in t, a similar rule would be needed, that we omit since it is perfectly analogous.

Decompose$_A$

$$\frac{A_f \in ax(f) \wedge C_f \notin ax(f) \wedge m \geq 2 \wedge n \geq m \wedge k \in \{1, \ldots, (n-m)+1\}}{\begin{array}{c}\langle f(t_1, \ldots, t_n) \stackrel{x}{=} f(s_1, \ldots, s_m) \wedge CT \mid S \mid \theta\rangle \Rightarrow \\ \langle f(t_1, \ldots, t_k) \stackrel{x_1}{=} s_1 \wedge f(t_{k+1}, \ldots, t_n) \stackrel{x_2}{=} f(s_2, \ldots, s_m) \wedge CT \mid S \mid \theta\sigma\rangle\end{array}}$$

where $\sigma = \{x \mapsto f(x_1, x_2)\}$, and x_1, x_2 are fresh variables

Fig. 6. Decomposition rule for an associative (non–commutative) function symbol f

Example 4. Let $f(f(a,c),b)$ and $f(c,b)$ be two terms where f is associative, i.e., $A_f \in ax(f)$. By applying the rules $Solve_E$, $Recover_E$, and $Decompose_A$ above, we end in a terminal configuration $\langle \emptyset \mid S \mid \theta\rangle$, where $\theta = \{x \mapsto f(x_3, b), x_4 \mapsto b\}$, thus we compute that the lgg modulo A of t and s is $f(x_3, b)$. The computation trace is shown in Figure 7.

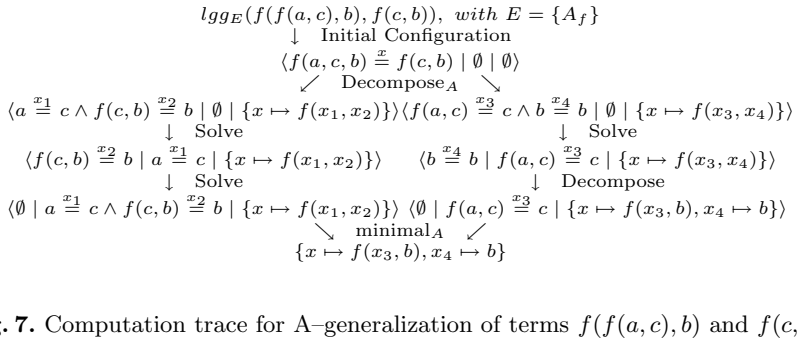

Fig. 7. Computation trace for A–generalization of terms $f(f(a,c),b)$ and $f(c,b)$).

4.4 Least general generalization modulo *AC*

In this section we provide a specific inference rule $Decompose_{AC}$ for handling function symbols obeying both the associativity and commutativity axioms. Note that we use again flattened versions of the terms, as in the associative case of Section 4.3. Actually, the new decomposition rule for the case AC is similar to the decompose inference rule for associative function symbols, except that all permutations of $f(t_1, \ldots, t_n)$ and $f(s_1, \ldots, s_m)$ are considered. Just notice that this rule is (don't know) non-deterministic, hence all possibilities must be explored.

Similarly to the rule $Decompose_A$, we give the rule $Decompose_{AC}$ for the case when, in the generalization problem $s \stackrel{x}{\triangleq} t$, the number of *alien terms* in s is greater than or equal to the number of alien terms in t. For the other way round, a similar rule would be needed, that we omit since it is perfectly analogous. To simplify, we write $\{i_1, \ldots, i_k\} \bar{\uplus} \{i_{k+1}, \ldots, i_n\} = \{1, \ldots, n\}$ to denote that for every $k, j \in \{1, \ldots, n\}$, $i_j \in \{1, \ldots, n\}$, $\{i_{k+1}, \ldots, i_n\} = \{1, \ldots, n\} \setminus \{i_1, \ldots, i_k\}$, $i_1 \leq \cdots \leq i_k$, and $i_{k+1} \leq \cdots \leq i_n$. Note that it is not necessary that $i_k \leq i_{k+1}$.

Decompose$_{AC}$

$$\frac{\{A_f, C_f\} \subseteq ax(f) \wedge n \geq m \ \wedge \{i_1, \ldots, i_{m-1}\} \uplus \{i_m, \ldots, i_n\} = \{1, \ldots, n\}}{\begin{array}{c}\langle f(t_1, \ldots, t_n) \stackrel{x}{=} f(s_1, \ldots, s_m) \wedge C \mid S \mid \theta\rangle \Rightarrow \\ \langle t_{i_1} \stackrel{x_1}{=} s_1 \wedge \ldots \wedge t_{i_{m-1}} \stackrel{x_{m-1}}{=} s_{m-1} \wedge f(t_{i_m}, \ldots, t_{i_n}) \stackrel{x_m}{=} s_m \wedge C \mid S \mid \theta\sigma\rangle\end{array}}$$

where $\sigma = \{x \mapsto f(x_1, \ldots, x_m)\}$, and $x_1, \ldots, x_m$ are fresh variables

Fig. 8. Decomposition rule for an associative–commutative function symbol f

Example 5. Let $t = f(a, f(a, b))$ and $s = f(f(b, b), a)$ be two terms where f is associative and commutative, i.e., $\{A_f, C_f\} \subseteq ax(f)$. By applying the rules $Solve_E$, $Recover_E$, and $Decompose_{AC}$ above, we end in two terminal configurations whose respective substitution components are $\theta_1 = \{x \mapsto f(x_1, x_1, x_3), x_2 \mapsto x_1\}$ and $\theta_2 = \{x \mapsto f(x_4, a, b), x_5 \mapsto a, x_6 \mapsto b\}$, thus we compute that the lggs modulo AC of t and s are $f(x_1, x_1, x_3)$ and $f(x_4, a, b)$. The corresponding computation trace is shown in Figure 9.

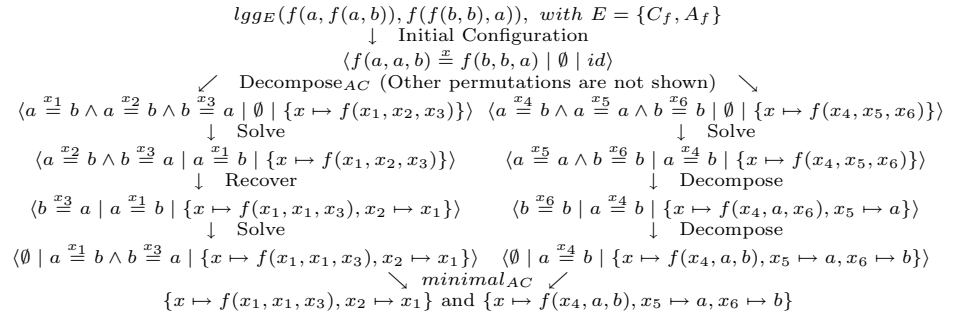

Fig. 9. Computation trace for AC–generalizations of terms $f(a, f(a, b))$ and $f(f(b, b), a)$.

4.5 Least general generalization modulo U

Finally, let us introduce the inference rule of Figure 10 for handling function symbols f which have an identity element e. This rule considers the identity axioms in a rather lazy or on-demand manner. The rule corresponds to the case when the root symbol f of the term t in the left–hand side of the constraint $t \stackrel{x}{=} s$ obeys the unity axioms. A companion rule for handling the case when the root symbol of the term s in the right–hand side obeys the unity axiom is omitted, that is perfectly analogous.

Example 6. Let $t = f(a, b, c, d)$ and $s = f(a, c)$ be two terms where $\{A_f, C_f, U_f\} \subseteq ax(f)$. By applying the rules $Solve_E$, $Recover_E$, $Decompose_{AC}$, and $Expand_U$ above, we end in a terminal configuration $\langle \emptyset \mid S \mid \theta\rangle$, where $\theta = \{x \mapsto f(a, f(c, f(x_5, x_6))), x_1 \mapsto a, x_2 \mapsto f(c, f(x_5, x_6)), x_3 \mapsto c, x_4 \mapsto f(x_5, x_6)\}$, thus we compute that the lgg modulo ACU of t and s is $f(a, c, x_5, x_6)$. The computation trace is shown in Figure 11.

Expand$_U$

$$\frac{U_f \in ax(f) \wedge root(t) \equiv f \wedge root(s) \not\equiv f \wedge s' \in \{f(e,s), f(s,e)\}}{\langle t \stackrel{x}{=} s \wedge CT \mid S \mid \theta\rangle \Rightarrow \langle t \stackrel{x}{=} s' \wedge CT \mid S \mid \theta\rangle}$$

Fig. 10. Inference rule for expanding function symbol f with identity element e

$$\begin{array}{c}
lgg_E(f(a,b,c,d), f(a,c)),\ with\ E = \{C_f, A_f, U_f\} \\
\downarrow \quad \text{Initial Configuration} \\
\langle f(a,b,c,d) \stackrel{x}{=} f(a,c) \mid \emptyset \mid id\rangle \\
\downarrow \quad \text{Decompose}_{AC}\ \text{(Other permutations are not shown)} \\
\langle a \stackrel{x_1}{=} a \wedge f(b,c,d) \stackrel{x_2}{=} c \mid \emptyset \mid \{x \mapsto f(x_1,x_2)\}\rangle \\
\downarrow \quad \text{Decompose} \\
\langle f(b,c,d) \stackrel{x_2}{=} c \mid \emptyset \mid \{x \mapsto f(a,x_2), x_1 \mapsto a\}\rangle \\
\downarrow \quad \text{Decompose}_U \\
\langle f(b,c,d) \stackrel{x_2}{=} f(c,e) \mid \emptyset \mid \{x \mapsto f(a,x_2), x_1 \mapsto a\}\rangle \\
\downarrow \quad \text{Decompose}_{AC}\ \text{(Other permutations are not shown)} \\
\langle c \stackrel{x_3}{=} c \wedge f(b,d) \stackrel{x_4}{=} e \mid \emptyset \mid \{x \mapsto f(a,f(x_3,x_4)), x_1 \mapsto a, x_2 \mapsto f(x_3,x_4)\}\rangle \\
\downarrow \quad \text{Decompose} \\
\langle f(b,d) \stackrel{x_4}{=} e \mid \emptyset \mid \{x \mapsto f(a,f(c,x_4)), x_1 \mapsto a, x_2 \mapsto f(c,x_4), x_3 \mapsto c\}\rangle \\
\downarrow \quad \text{Decompose}_U \\
\langle f(b,d) \stackrel{x_4}{=} f(e,e) \mid \emptyset \mid \{x \mapsto f(a,f(c,x_4)), x_1 \mapsto a, x_2 \mapsto f(c,x_4), x_3 \mapsto c\}\rangle \\
\downarrow \quad \text{Decompose}_{AC}\ \text{(Other permutations are not shown)} \\
\langle b \stackrel{x_5}{=} e \wedge d \stackrel{x_6}{=} e \mid \emptyset \mid \{x \mapsto f(a,f(c,f(x_5,x_6))), x_1 \mapsto a, x_2 \mapsto f(c,f(x_5,x_6)), x_3 \mapsto c, x_4 \mapsto f(x_5,x_6)\}\rangle \\
\downarrow \quad \text{Solve} \\
\langle d \stackrel{x_6}{=} e \mid b \stackrel{x_5}{=} e \mid \{x \mapsto f(a,f(c,f(x_5,x_6))), x_1 \mapsto a, x_2 \mapsto f(c,f(x_5,x_6)), x_3 \mapsto c, x_4 \mapsto f(x_5,x_6)\}\rangle \\
\downarrow \quad \text{Solve} \\
\langle \emptyset \mid b \stackrel{x_5}{=} e \wedge d \stackrel{x_6}{=} e \mid \{x \mapsto f(a,f(c,f(x_5,x_6))), x_1 \mapsto a, x_2 \mapsto f(c,f(x_5,x_6)), x_3 \mapsto c, x_4 \mapsto f(x_5,x_6)\}\rangle \\
\downarrow \quad \text{minimal}_{ACU} \\
\{x \mapsto f(a,f(c,f(x_5,x_6))), x_1 \mapsto a, x_2 \mapsto f(c,f(x_5,x_6)), x_3 \mapsto c, x_4 \mapsto f(x_5,x_6)\}
\end{array}$$

Fig. 11. Computation trace for U–generalization of terms $f(a,b,c,d)$ and $f(a,c)$.

4.6 A general ACU-generalization method and its correctness

For the general case when different function symbols satisfying different associativity and/or commutativity and/or identity axioms are considered, we can use the rules above all together, with no need whatsoever for any changes or adaptations.

The key property of all the above inference rules is their *locality*: they are local to the given top function symbol in the left term (or right term in some cases) of the constraint they are acting upon, irrespective of what other function symbols and what other axioms may be present in the given signature Σ and theory E. Such a locality means that these rules are *modular*, in the sense that they do not need to be changed or modified when new function symbols are added to the signature and new A, and/or C, and/or U axioms are added to E. However, when new axioms are added to E, some rules that applied before (for example decomposition for an f which before satisfied $ax(f) = \emptyset$, but now has $ax(f) \neq \emptyset$) may not apply, and, conversely, some rules that did not apply before now may apply (because new axioms are added to f). But *the rules themselves do not change!* They are the same and can be used to compute the set of lggs of two terms modulo *any* theory E in the *parametric* family $\mathbb{E}$ of theories of the form $E = \bigcup_{f \in \Sigma} ax(f)$, where $ax(f) \subseteq \{A_f, C_f, U_f\}$.

Termination of the transition system $(Conf, \Rightarrow)$ is straightforward. Let us prove the correctness and completeness of our algorithm in a modular way. First, we need some auxiliary lemmata. The following result ensures that the syntactic lgg of two terms, which is computed by the inference rules of Figure 1, can be also obtained by $\Rightarrow$, i.e., the smallest transition relation satisfying the inference rules of Figures 3, 4, 6, 8, and 10.

Lemma 1. *Given terms t and s, an equational theory $E \in \mathbb{E}$, and a fresh variable x, $u \in lgg(t, s)$ (i.e., without applying any equation in E) if and only if $\langle t \stackrel{x}{\triangleq} s \mid \emptyset \mid id \rangle \Rightarrow^* \langle \emptyset \mid S \mid \theta \rangle$ for some S and θ, and there is a renaming ρ s.t. $u\rho = x\theta$.*

The following lemma is immediate by Theorem 2 and confluence of the rewrite theory $\overleftrightarrow{E}$. Given a regular equational theory E, we write $\overleftrightarrow{E} = \{l \to r,\ r \to l \ \mid \ l = r \ \in E\}$, where rules in both directions are indeed rewrite rules, provided we also allow left-hand sides consisting of a single variable. It is then easy to show that $\overleftrightarrow{E}$ is confluent. Note that all equational theories E in our parametric family $\mathbb{E}$ of theories are regular.

Lemma 2. *Given terms t and s and an equational theory $E \in \mathbb{E}$, $u \in lgg_E(t, s)$ implies that there exist v, v' s.t. $t \to^n_{\overleftrightarrow{E}} v$, $s \to^m_{\overleftrightarrow{E}} v'$, and $u = lgg(v, v')$. Moreover, there exists at least one* minimal *pair (n_0, m_0) for the computation of u, i.e., minimal in the sense of avoiding equation applications on t or s that do not have any effect in the computation of u, such as applying an equation and rolling it back later or equation applications at the bindings of the subtitutions μ, μ' such that $u\mu =_E t$ and $u\mu' =_E s$.*

Furthermore, by confluence of the rewrite theory $\overleftrightarrow{E}$, we can reorder the two rewrite sequences $t \to^{n_0}_{\overleftrightarrow{E}} v$ and $s \to^{m_0}_{\overleftrightarrow{E}} v'$ in the following form (similar for s and v') $t \stackrel{p_1}{\to}_{\overleftrightarrow{E}} t_1 \cdots \stackrel{p_n}{\to}_{\overleftrightarrow{E}} v$ such that $p_1 \leq p_2 \leq \cdots \leq p_{n_0}$.

We also prove that decomposing terms is harmless whenever no equation of E is applied at the root position.

Lemma 3. *Given terms $t = f(t_1, \ldots, t_n)$ and $s = f(s_1, \ldots, s_n)$ in an equational theory $E \in \mathbb{E}$, and $u \in lgg_E(t, s)$, if there exist terms t', s', v, v' such that $t \xrightarrow[E]{p_1} t' \cdots \xrightarrow[E]{p_n} v$, $\Lambda < p_i$ for $1 \leq i \leq n$, $s \xrightarrow[E]{p'_1} s' \cdots \xrightarrow[E]{p'_m} v'$, $\Lambda < p'_i$ for $1 \leq i \leq m$, and $u = lgg(v, v')$, then $u = f(u_1, \ldots, u_n)$, and we have that for $1 \leq i \leq n$, $u_i \in lgg_E(t_i, s_i)$.*

We prove completeness of $\Rightarrow$.

Lemma 4. *Given terms t and s, an equational theory $E \in \mathbb{E}$, and a fresh variable x, $u \in lgg_E(t, s)$ if there exist S and θ s.t. $\langle t \overset{x}{\triangleq} s \mid \emptyset \mid id \rangle \Rightarrow^* \langle \emptyset \mid S \mid \theta \rangle$ and there is a renaming ρ s.t. $u\rho =_E x\theta$.*

Proof. By Lemma 2, there exist v, v' s.t. $t \rightarrow^n_{\underset{E}{\leftrightarrow}} v$, $s \rightarrow^m_{\underset{E}{\leftrightarrow}} v'$, and $u = lgg(v, v')$, where the pair (n, m) is minimal. We reason by induction on such minimal pair (n, m) of applications of the equations in E.

- ($n = 0$ and $m = 0$) Immediate by Lemma 1 and Theorem 1.
- ($n > 0$ or $m > 0$) Let $t \xrightarrow[E]{p} t' \rightarrow^{n-1}_{\underset{E}{\leftrightarrow}} v$ and $s \xrightarrow[E]{q} s' \rightarrow^{m-1}_{\underset{E}{\leftrightarrow}} v'$. Let us assume without loss of generality that $p \leq q$. Now, we consider only the case for t, since the case for s is perfectly analogous. By Lemma 2, for each $p' < p$, $root(t|_{p'}) = root(s|_{p'})$. Let $root(t|_p) = f$. We consider whether there are rewrite steps in $t' \rightarrow^{n-1}_{\underset{E}{\leftrightarrow}} v$ at position p or not.
  - If there are still steps at position p, then it may be possible that $root(u|_p) \neq root(t|_p)$. By induction hypothesis, $u \in lgg_E(t', s)$ and there are S, θ, and a renaming ρ s.t. $\langle t' \overset{x}{\triangleq} s \mid \emptyset \mid id \rangle \Rightarrow^* \langle \emptyset \mid S \mid \theta \rangle$ and $u\rho =_E x\theta$. Note that $ax(root(t|_p)) \neq \emptyset$. However, we have $\langle t \overset{x}{\triangleq} s \mid \emptyset \mid id \rangle \Rightarrow^* \langle \emptyset \mid S \mid \theta \rangle$ and $u\rho =_E x\theta$, since we can easily prove that, for all the cases, the effect of the rewrite step $\xrightarrow[E]{p}$ is mimicked by the inference rules of Figures 4, 6, 8, and 10.
  - Now, we consider the case when there is no rewrite step at position p. By induction hypothesis, $u \in lgg_E(t', s)$ and there are S, θ, and a renaming ρ s.t. $\langle t' \overset{x}{\triangleq} s \mid \emptyset \mid id \rangle \Rightarrow^* \langle \emptyset \mid S \mid \theta \rangle$ and $u\rho =_E x\theta$. Let us assume without loss of generality that there is no rewrite step at position p in the rewrite sequence $s' \rightarrow^{m-1}_{\underset{E}{\leftrightarrow}} v'$. By Lemma 3, let $t'|_p = f(t'_1, \ldots, t'_n)$, $s|_p = f(s_1, \ldots, s_n)$ and $u|_p = f(u_1, \ldots, u_n)$, we have that for $1 \leq i \leq n$, $u_i = lgg_E(t_i, s_i)$. Finally, we have $\langle t \overset{x}{\triangleq} s \mid \emptyset \mid id \rangle \Rightarrow^* \langle \emptyset \mid S \mid \theta \rangle$ and $u\rho =_E x\theta$, since the rules of Figures 4, 6, and 8 mimick the effect of both the rewrite step $\xrightarrow[E]{p}$ and the term decomposition. Note that for the case of $U_f \in ax(f)$, both the equation application and the decomposition are mimicked, respectively, by the inference rule of Figure 10 and the corresponding decompose rule of the rules of Figures 4, 6, and 8. □

Theorem 3. *Given terms t and s, an equational theory $E \in \mathbb{E}$, and a fresh variable x, then $lgg_E(t, s) = minimal_{<_E}(\{x\theta \mid \langle t \overset{x}{\triangleq} s \mid \emptyset \mid id \rangle \Rightarrow^* \langle \emptyset \mid S \mid \theta \rangle\})$, up to renaming.*

Proof. By Lemma 4 and minimality of $minimal_{<_E}$. □

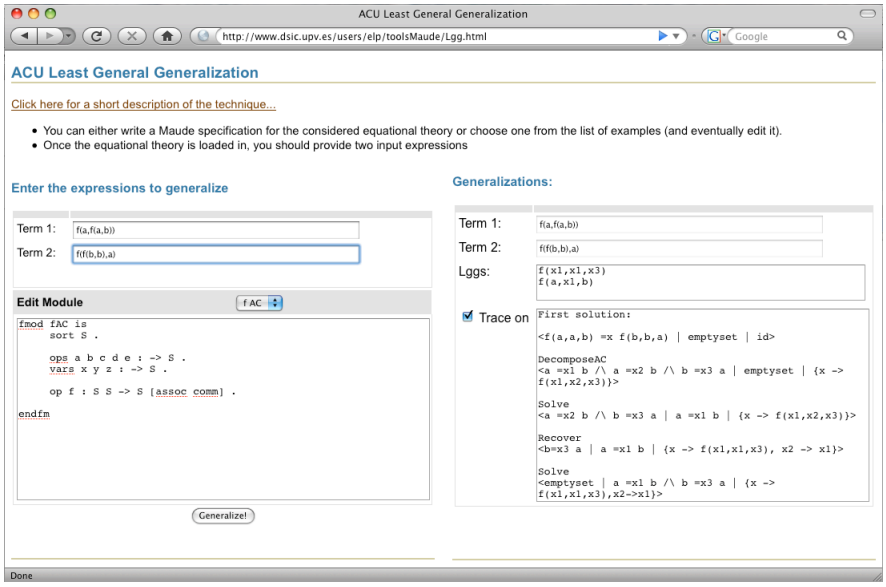

Fig. 12. Snapshot of the equational lgg web interface

5 Implementation

The ACU least general generalization algorithm presented here has been implemented in Maude [8], with a Web interface written in Java. The core of the tool contains about 200 lines of Maude code. A snapshot of the Web tool is shown in Figure 12. The Web tool is publicly available together with a set of examples at the following url: `http://www.dsic.upv.es/users/elp/toolsMaude/Lgg.html`.

6 Conclusions and Future Work

We have presented a modular equational generalization algorithm that computes a minimal and complete set of least general generalizations for two terms modulo any combination of associativity, commutativity and identity axioms for the binary symbols in the theory. Our algorithm is directly applicable to any untyped declarative languages and reasoning systems. However, it would be highly desirable to support generalization modulo equational theories (Σ, E) where Σ is a typed signature such as for example an order–sorted signature, since a number of rule–based languages such as ASF+SDF [5], Elan [6], OBJ [15], CafeOBJ [11], and Maude [8] support order–sorted or many–sorted signatures. All existing generalization algorithms, with the exception of the work of Pfenning on generalization in the higher-order setting of the calculus of constructions [23], assume an untyped setting. However, the algorithm for generalization in the calculus of constructions of [23] cannot be used for order-sorted theories. In [1], we have developed an order–sorted generalization algorithm for the case where the set E of axioms is empty. It would be very useful to combine the order–sorted and

the E–generalization inference systems into a single calculus supporting both types and equational axioms. However, this combination seems to us non–trivial and is left for future work.

In our own work, we plan to use the above–mentioned order-sorted equational generalization algorithm as a key component of a narrowing-based partial evaluator (PE) for programs in order-sorted rule-based languages such as OBJ, CafeOBJ, and Maude. This will make available for such languages useful narrowing–driven PE techniques developed for the untyped setting in, e.g., [2,3]. We are also considering adding this generalization mechanism to an inductive theorem prover such a Maude's ITP [9] to support automatic conjecture of lemmas. This will provide a typed analogue of similar automatic lemma conjecture mechanisms in untyped inductive theorem provers such as Nqthm [7] and its ACL2 successor [17].

References

1. M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda. Oder–Sorted Generalization. Technical Report DSIC-II/5/08, DSIC-UPV, 2008. Submitted for publication.
2. M. Alpuente, M. Falaschi, and G. Vidal. Partial evaluation of functional logic programs. *ACM Trans. Program. Lang. Syst.*, 20(4):768–844, 1998.
3. M. Alpuente, S. Lucas, M. Hanus, and G. Vidal. Specialization of functional logic programs based on needed narrowing. *TPLP*, 5(3):273–303, 2005.
4. F. Baader and W. Snyder. Unification theory. In *Handbook of Automated Reasoning*. Elsevier, 1999.
5. J.A. Bergstra, J. Heering, and P. Klint. *Algebraic Specification*. ACM Press, 1989.
6. P. Borovanský, C. Kirchner, H. Kirchner, and P.-E. Moreau. ELAN from a rewriting logic point of view. *Theoretical Computer Science*, 285:155–185, 2002.
7. R. Boyer and J. Moore. *A Computational Logic*. Academic Press, 1980.
8. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude - A High-Performance Logical Framework*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
9. M. Clavel and M. Palomino. The ITP tool's manual. Universidad Complutense, Madrid, April 2005, http://maude.sip.ucm.es/itp/.
10. V. Cortier, S. Delaune, and P. Lafourcade. A Survey of Algebraic Properties used in Cryptographic Protocols. *Journal of Computer Security*, 14(1):1–43, 2006.
11. R. Diaconescu and K. Futatsugi. *CafeOBJ Report*, volume 6 of *AMAST Series in Computing*. World Scientific, AMAST Series, 1998.
12. S. Escobar, J. Meseguer, and P. Thati. Narrowing and rewriting logic: from foundations to applications. *Electr. Notes Theor. Comput. Sci.*, 177:5–33, 2007.
13. Santiago Escobar, Catherine Meadows, and José Meseguer. A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties. *Theoretical Computer Science*, 367(1-2):162–202, 2006.
14. J. P. Gallagher. Tutorial on specialisation of logic programs. In *PEPM '93: Proceedings of the 1993 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pages 88–98, New York, NY, USA, 1993. ACM.
15. J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In *Software Engineering with OBJ: Algebraic Specification in Action*, pages 3–167. Kluwer, 2000.
16. G. Huet. *Resolution d'Equations dans des Langages d'Order 1, 2,...,ω*. PhD thesis, Univ. Paris VII, 1976.

17. M. Kaufmann, P. Manolios, and J.S Moore. *Computer-Aided Reasoning: An Approach*. Kluwer, 2000.
18. J.-L. Lassez, M. J. Maher, and K. Marriott. Unification Revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Los Altos, Ca., 1988.
19. J. Meseguer. Conditioned rewriting logic as a united model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.
20. J. Meseguer. Membership algebra as a logical framework for equational specification. In *WADT*, pages 18–61, 1997.
21. T. Æ. Mogensen. Glossary for partial evaluation and related topics. *Higher-Order and Symbolic Computation*, 13(4), 2000.
22. S. Muggleton. Inductive Logic Programming: Issues, Results and the Challenge of Learning Language in Logic. *Artif. Intell.*, 114(1-2):283–296, 1999.
23. F. Pfenning. Unification and anti-unification in the calculus of constructions. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 74–85. IEEE Computer Society, 1991.
24. G.D. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.
25. G.D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.
26. R.J. Popplestone. An experiment in automatic induction. In *Machine Intelligence*, volume 5, pages 203–215. Edinburgh University Press, 1969.
27. J. Reynolds. Transformational systems and the algebraic structure of atomic formulas. *Machine Intelligence*, 5:135–151, 1970.
28. J.H. Siekmann. Unification Theory. *Journal of Symbolic Computation*, 7:207–274, 1989.
29. TeReSe, editor. *Term Rewriting Systems*. Cambridge University Press, Cambridge, 2003.