Alpuente Frasnedo, M.; Escobar Román, S.; Espert Real, J.; Meseguer, J. (2014). A Modular Order-sorted Equational Generalization Algorithm. Information and Computation. 235:98-136. doi:10.1016/j.ic.2014.01.006.

A Modular Order-sorted Equational Generalization Algorithm

María Alpuente[a], Santiago Escobar[a,*], Javier Espert[a], José Meseguer[b]

[a] *Universitat Politècnica de València, Spain*
[b] *University of Illinois at Urbana-Champaign, USA*

Abstract

Generalization, also called anti-unification, is the dual of unification. Given terms t and t', a generalizer is a term t'' of which t and t' are substitution instances. The dual of a most general unifier (mgu) is that of least general generalizer (lgg). In this work, we extend the known untyped generalization algorithm to, first, an order-sorted typed setting with sorts, subsorts, and subtype polymorphism; second, we extend it to work modulo equational theories, where function symbols can obey any combination of associativity, commutativity, and identity axioms (including the empty set of such axioms); and third, to the combination of both, which results in a modular, order-sorted equational generalization algorithm. Unlike the untyped case, there is in general no single lgg in our framework, due to order-sortedness or to the equational axioms. Instead, there is a finite, minimal and complete set of lggs, so that any other generalizer has at least one of them as an instance. Our generalization algorithms are expressed by means of inference systems for which we give proofs of correctness. This opens up new applications to partial evaluation, program synthesis, and theorem proving for typed equational reasoning systems and typed rule-based languages such as ASF+SDF, Elan, OBJ, Cafe-OBJ, and Maude.

1. Introduction

Generalization is a formal reasoning component of many symbolic frameworks, including theorem provers, and automatic techniques for program analysis, synthesis, verification, compilation, refactoring, test case generation, learning, specialisation, and transformation; see, *e.g.,* (Boyer and Moore, 1980; Buly-

☆M. Alpuente, S. Escobar, and J. Espert have been partially supported by the EU (FEDER) and the Spanish MEC/MICINN under grant TIN 2010-21062-C02-02, and by Generalitat Valenciana PROMETEO2011/052. J. Meseguer has been supported by NSF Grants CNS 09-04749, and CCF 09-05584.

*Corresponding author

Email addresses: alpuente@dsic.upv.es (María Alpuente), sescobar@dsic.upv.es (Santiago Escobar), jespert@dsic.upv.es (Javier Espert), meseguer@cs.uiuc.edu (José Meseguer)

chev et al., 2010; Gallagher, 1993; Kitzelmann and Schmid, 2006; Kutsia et al., 2011; Lu et al., 2000; Muggleton, 1999; Pfenning, 1991). Generalization, also called anti-unification, is the dual of unification. Given terms t and t', a generalizer of t and t' is a term t'' of which t and t' are substitution instances. The dual of a most general unifier (mgu) is that of a least general generalizer (lgg), that is, a generalizer that is more specific than any other generalizer. Whereas unification produces most general unifiers that, when applied to two expressions, instantiate the inputs to make them equivalent to the most general common instance (Lassez et al., 1988), generalization abstracts the inputs by computing their most specific generalizer. As in unification, where the mgu is of interest, in the sequel we are interested in the lgg or, as we shall see for the order-sorted, equational case treated in this article, 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 (Alpuente et al., 1995; Baader and Snyder, 1999).

As an important application, generalization is a relevant component for ensuring termination of program manipulation techniques such as automatic program analysis, synthesis, specialisation and transformation, in automatic theorem proving, logic programming, typed lambda calculus, term rewriting, *etc.* For instance, in the partial evaluation (PE) of logic programs (Gallagher, 1993), the general idea is to construct a set of finite (possibly partial) deduction trees for a set of initial function calls (*i.e.,* generic function calls using logical variables), and then extract from those trees a new program P that allows any instance of the initial calls to be executed. To ensure that the partially evaluated program P *covers* all the possible initial function calls, most PE procedures recursively add other function calls that show up dynamically during the process of constructing the deduction trees for the set of calls to be specialized. This process could go on forever by adding more and more function calls that have to be specialized and thus it 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, then both calls are generalized by computing their lgg and the specialization process is restarted from the generalized call, ensuring that both calls will be covered by the new resulting partially evaluated program.

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 (Muggleton, 1999), and also to conjecture lemmas in inductive theorem provers such as Nqthm (Boyer and Moore, 1980) and its ACL2 extension (Kaufmann et al., 2000). In the literature on machine learning and partial evaluation, least general generalization is also known as *most specific generalization* (msg) and *least common anti–instance* (lcai) (Mogensen, 2000). Least general generalization was originally introduced by Plotkin in (Plotkin, 1970), see also (Reynolds, 1970). Indeed, Plotkin's work originated from the consideration in (Popplestone, 1969) that, since unification is useful in automatic deduction by the resolution method, its dual might prove helpful for induction. Generalization is also used in test case generation techniques to achieve appropriate coverage

(Belli and Jack, 1998). Applications of generalization to invariant generation and software clone detection are described in (Bulychev et al., 2010). Suggestion for auxiliary lemmas in equational inductive proofs, computation of construction laws for given term sequences, and learning of screen editor command sequences by using generalization are discussed in (Burghardt, 2005).

To the best of our knowledge, most previous generalization algorithms assume an *untyped setting*; two notable exceptions are the generalization in the higher-order setting of the calculus of constructions of (Pfenning, 1991) and the order-sorted feature term generalization of (Aït-Kaci, 1983; Aït-Kaci and Sasaki, 2001). However, many applications, for example to partial evaluation, theorem proving, and program learning, for typed rule-based languages such as ASF+SDF (Bergstra et al., 1989), Elan (Borovanský et al., 2002), OBJ (Goguen et al., 2000), CafeOBJ (Diaconescu and Futatsugi, 1998), and Maude (Clavel et al., 2007), require a first-order typed version of generalization which does not seem to be available: we are not aware of any existing algorithm. Moreover, several of the above-mentioned languages have an expressive *order-sorted* typed setting with sorts, subsorts (where subsort inclusions form a partial order and are interpreted semantically as set-theoretic inclusions of the corresponding data sets), and subsort-overloaded function symbols (a feature also known as *subtype polymorphism*), so that a symbol, for example +, can simultaneously exist for various sorts in the same subsort hierarchy, such as + for natural, integers, and rationals, and its semantic interpretations agree on common data items. Because of its support for *order-sorted* specifications, our generalization algorithm can be applied to generalization problems in all the above-mentioned rule-based languages.

Also, quite often all the above mentioned 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 (Bergstra et al., 1989), Elan (Borovanský et al., 2002), OBJ (Goguen et al., 2000), CafeOBJ (Diaconescu and Futatsugi, 1998), and Maude (Clavel et al., 2007) some function symbols may be declared to obey given algebraic laws (the so-called *equational attributes* of associativity and/or commutativity and/or identity in 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. Again, our generalization algorithm applies to all such typed languages and theorem provers because of its support for associativity and/or commutativity and/or identity axioms.

Surprisingly, unlike order-sorted unification, equational unification, and order-sorted equational unification, which all three have been thoroughly investigated in the literature —see, *e.g.,* (Baader and Snyder, 1999; Meseguer et al., 1989; Schmidt-Schauss, 1986; Siekmann, 1989; Smolka et al., 1989)— to the best of our knowledge there seems to be no previous, systematic treatment of order-sorted generalization, equational generalization, and order-sorted equa-

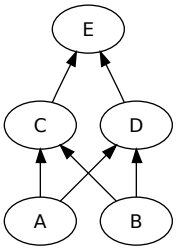

Figure 1: Sort hierarchy

tional generalization, although some order-sorted cases and some unsorted equational cases have been studied (see below).

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 $t = x$ trivially generalizes the two terms, with x being a variable. Another possible generalizer 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 generalizer* (up to variable renaming). Moreover, if we have order-sorted information as shown in Figure 1 such that constant a is of sort A, constant b is of sort B, but symbol f has two typings $f : \mathsf{C} \times \mathsf{C} \to \mathsf{C}$ and $f : \mathsf{D} \times \mathsf{D} \to \mathsf{D}$, where A and B are subsorts of C and D, then there are eight least general generalizers $f(f(x{:}\mathsf{C}, x{:}\mathsf{C}), y{:}\mathsf{C})$, $f(f(x{:}\mathsf{C}, x{:}\mathsf{C}), y{:}\mathsf{D})$, $f(f(x{:}\mathsf{C}, x{:}\mathsf{D}), y{:}\mathsf{C})$, $f(f(x{:}\mathsf{C}, x{:}\mathsf{D}), y{:}\mathsf{D})$, $f(f(x{:}\mathsf{D}, x{:}\mathsf{C}), y{:}\mathsf{C})$, $f(f(x{:}\mathsf{D}, x{:}\mathsf{C}), y{:}\mathsf{D})$, $f(f(x{:}\mathsf{D}, x{:}\mathsf{D}), y{:}\mathsf{C})$, and $f(f(x{:}\mathsf{D}, x{:}\mathsf{D}), y{:}\mathsf{D})$. Note that the variable identifier x is given two different sorts C and D in some of the generalizers, which actually stands for two different variables $x{:}\mathsf{C}$ and $x{:}\mathsf{D}$ due to sort incomparability. Hence, we get two generalizers $f(f(x{:}\mathsf{C}, x{:}\mathsf{C}), y{:}\mathsf{C})$ and $f(f(x{:}\mathsf{C}, x{:}\mathsf{D}), y{:}\mathsf{C})$, which are both correct and incomparable due to the sort structure. If we have equational properties for symbol f, for instance f being associative and commutative, and we disregard order-sorted information, then there are two least general generalizers $f(f(x,x),y)$ and $f(f(a,b),y)$, which are incomparable using associativity and commutativity. Finally, if we combine order-sorted information and the same algebraic properties, then there are eight least general generalizers, six from the eight generalizers showed above, $f(f(x{:}\mathsf{C}, x{:}\mathsf{C}), y{:}\mathsf{C})$, $f(f(x{:}\mathsf{C}, x{:}\mathsf{C}), y{:}\mathsf{D})$, $f(f(x{:}\mathsf{C}, x{:}\mathsf{D}), y{:}\mathsf{C})$, $f(f(x{:}\mathsf{C}, x{:}\mathsf{D}), y{:}\mathsf{D})$, $f(f(x{:}\mathsf{D}, x{:}\mathsf{D}), y{:}\mathsf{C})$, and $f(f(x{:}\mathsf{D}, x{:}\mathsf{D}), y{:}\mathsf{D})$, plus two extra generalizers coming from the unsorted generalizer $f(f(a,b),y)$ showed above, $f(a.\mathsf{A}, b.\mathsf{B}, y{:}\mathsf{C})$ and $f(a.\mathsf{A}, b.\mathsf{B}, y{:}\mathsf{D})$.

The extension of the generalization algorithm to deal with order-sorted functions and equational theories is nontrivial, because of two important reasons. First, as already mentioned and illustrated in the above example, the existence and uniqueness of a least general generalizer is typically lost. There is a finite

and minimal set of *least general generalizers* for two terms, so that any other generalizer has at least one of those as an instance. Such a set of lggs is the dual analogue of a minimal and complete set of unifiers for non-unitary unification algorithms, such as those for order-sorted unification, *e.g.,* (Meseguer et al., 1989; Schmidt-Schauss, 1986; Smolka et al., 1989), and for equational unification, see, *e.g.,* (Baader and Snyder, 1999; Siekmann, 1989). Second, similarly to the case of equational unification (Siekmann, 1989), computing least general generalizers modulo an equational theory E is a difficult task due to the possibility of combinatorial explosion. Depending on the theory E, a generalization problem may be undecidable, and even if it is decidable, may have infinitely many solutions.

This article develops several generalization algorithms: an *order-sorted* generalization algorithm, a *modular* E–generalization algorithm, and the combined version of both algorithms into an order-sorted and modular E-generalization algorithm. In this article, we do not address the E-generalization problem in its fullest generality. Our modular E–generalization algorithm works for a *parametric* family of theories (Σ, E) such that any binary function symbol $f \in \Sigma$ can have any combination of the following axioms: (i) *associativity* (A_f) $f(x, f(y, z)) \doteq f(f(x, y), z)$; (ii) *commutativity* (C_f) $f(x, y) \doteq f(y, x)$, and (iii) *identity* (U_f) for f of a constant symbol, say, e, *i.e.,* $f(x, e) \doteq x$ and $f(e, x) \doteq x$. In particular, f may not satisfy any such axioms, which when it happens for all binary symbols $f \in \Sigma$ gives us the standard, syntactic (order-sorted) generalization algorithm as a special case. As it is usual in current treatments of different formal deduction mechanisms, and has become standard for the dual case of unification algorithms since Martelli and Montanari (Jouannaud and Kirchner, 1991; Martelli and Montanari, 1982), we specify each generalization process by means of an inference system rather than by an imperative-style algorithm.

Our contribution and plan of the paper

After some preliminaries in Section 2, we recall in Section 3 a syntactic unsorted generalization algorithm as a special case to motivate later extensions. The main contributions of the paper can be summarized as follows:

1. An order-sorted generalization algorithm (in Section 4). If two terms are related in the sort ordering (their sorts are both in the same connected component of the partial order of sorts), then there is in general no single lgg, but the algorithm computes a finite, complete and minimal set of *least general generalizers*, so that any other generalizer has at least one of those as an instance. Such a set of lggs is the dual analogue of a minimal and complete set of unifiers for non-unitary unification algorithms, such as those for order-sorted unification.
2. A modular equational generalization algorithm (in Section 5). Indeed, we provide different generalization algorithms —one for each kind of equational axiom— but the overall 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, *e.g.,* (Baader and Snyder, 1999). To the best of our knowledge, ours are the first equational least general generalization algorithms in the literature. An interesting result is that associative generalization is finitary, whereas associative unification is infinitary, see *e.g.,* (Baader and Snyder, 1999).

3. An order-sorted modular equational generalization algorithm (in Section 6), which combines and refines the inference rules given in Sections 4 and 5.
4. Formal correctness, completeness, and termination results for all the above generalization algorithms.
5. In Section 7, we present an implementation of the order-sorted, modular equational generalization algorithm, which is publicly available in the Maude system, followed by some conclusions and directions for future work in Section 8.

This paper is an extended and improved version of (Alpuente et al., 2009a,b) which unifies both, the order-sorted generalization of (Alpuente et al., 2009b) and the equational generalization of (Alpuente et al., 2009a) into a novel and more powerful, combined algorithm. The proposed algorithms should be of interest to developers of rule-based languages, theorem provers and equational reasoning programs, as well as program manipulation tools such as program analyzers, partial evaluators, test case generators, and machine learning tools, for (order-sorted) declarative languages and reasoning systems supporting commonly occurring equational axioms such as associativity, commutativity and identity in a built–in and efficient way. For instance, this includes many theorem provers, and a variety of rule-based languages such as ASF+SDF, OBJ, CafeOBJ, Elan, and Maude. Since the many-sorted and unsorted settings are special instances of the order-sorted case, our algorithm applies *a fortiori* to those less expressive settings.

Related work

Generalization goes back to work of Plotkin (Plotkin, 1970), Reynolds (Reynolds, 1970), and Huet (Huet, 1976) and has been studied in detail by other authors; see for example the survey (Lassez et al., 1988). Plotkin (Plotkin, 1970) and Reynolds (Reynolds, 1970) gave imperative–style algorithms for generalization, which are both essentially the same. Huet's generalization algorithm (Huet, 1976), formulated as a pair of recursive equations, cannot be understood as an automated calculus due to some implicit assumptions in the treatment of variables. A deterministic reconstruction of Huet's algorithm is given in (Østvold, 2004) which does not consider types either. A many-sorted generalization algorithm was presented in (Frisch and Page, 1990) that is provided with the so-called S-sentences, which can be seen as a logical notation for encoding taxonomic (or ordering) information. Anti-unification for unranked terms,

which differ from the standard ones by not having fixed arity for function symbols, and for finite sequences of such terms (called hedges) is investigated in (Kutsia et al., 2011); efficiency of the algorithm is improved by imposing a rigidity function that is a parameter of the improved algorithm. The algorithm for higher-order generalization in the calculus of constructions of (Pfenning, 1991) does not consider order-sorted theories or equational axioms either, and for any two higher-order patterns, either there is no lgg (because the types are incomparable), or there is a unique lgg.

The significance of equational generalization was already pointed out by Pfenning in (Pfenning, 1991): "*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*". However, to the best of our knowledge, we are not aware of any existing equational generalization algorithm modulo the combination of associativity, commutativity and identity axioms. Actually, equational generalization has been absolutely neglected, except for the theory of associativity and commutativity (Pottier, 1989) (in french) and for commutative theories (Baader, 1991). For the commutative case, (Baader, 1991) shows that all commutative theories are of generalization type 'unitary', but no generalization algorithm is provided. Pottier (Pottier, 1989) provides (unsorted) inference rules which mechanize generalization in AC theories, but these rules do not apply to the separate cases of C or A alone, nor to arbitrary combinations of the C, A, and U axioms. Finally, (Burghardt, 2005) presented a specially tailored algorithm that uses grammars to compute a finite representation of the (usually infinite) set of all E-generalizers of given terms, provided that E leads to regular congruence classes, which happens when E is the deductive closure of finitely many ground equations. However, as a natural consequence of representing equivalence classes of terms as regular tree grammars, the result of the E-generalization process is not a term, but a regular tree grammar of terms.

Least general generalization in an order-sorted typed setting was first investigated in (Aït-Kaci, 1983). A generalization algorithm is proposed in (Aït-Kaci, 1983) for feature terms, which are sorted, possibly nested, attribute-based structures which extend algebraic terms by relaxing the fixed arity and fixed indexing constraints. Feature terms generalize first-order terms by providing a natural way to describe incomplete information. This is done by adding *features* (or attribute labels) to a sort as argument indicators. In other words, parameters are identified by name (regardless of their order or position). Feature terms (previously known as indexed terms or Ψ-terms) were originally proposed as flexible record structures for logic programming and then used to describe different data models, including attributed typed objects, in rule-based languages which are oriented towards applications to knowledge representation and natural language processing.

Since functor symbols of feature terms are ordered sorts, a feature term can be thought of as a type template which represents a set-denoting sort. By choosing to define types to be terms, and the type classification ordering to be

term instantiation, the resulting type system is a lattice whose *meet* operation (*i.e.,* greatest lower bound) w.r.t. the subsumption relation induced by the subset ordering (term instantiation) is first-order unification, and whose *join* operation (*i.e.,* least upper bound) is first-order generalization. This model is familiar to Prolog programmers though unlike any other type system available in typed languages. Moreover, by considering a partial order on functors, the set of sorts is also given a pre-order structure. Intuitively, a feature term S is subsumed by a feature term T if S contains more information than T, or, equivalently, S denotes a subset of T. Under this subsumption order, the set of all feature terms is a prelattice provided the sort symbols are ordered as a lattice. Generalization is then defined as computing greater lower bounds in the prelattice of feature terms. The lgg of feature terms is also described in (Plaza, 1995) and a practical (yet incomplete) implementation is provided in (Armengol, 1998; Armengol and Plaza, 2000). We also refer to (Plaza, 1995) for an account of several variants of feature descriptions, as used in computational linguistics and related areas, where generalization is recast as the retrieval of common structural similarity.

A rich description level is achieved when types are viewed as constraints. In this context, terms can be thought of as "crystallized" syntaxes that dissolve into a semantically equivalent conjunction of elementary constraints, best defined as a "soup," thanks to conjunction being associative and commutative. In the constraint setting, feature terms correspond to order-sorted feature (OSF) constraints in solved form (a normal form). Generalization in the OSF foundation is investigated in (Aït-Kaci and Sasaki, 2001), where an axiomatic definition of feature term generalization is provided, together with its operational realization. In the axiomatic definition, generalization is presented as an OSF-constraint construction process: the information conveyed by OSF terms is given an alternative, syntactic presentation by means of a constraint clause, and generalization is then defined by means of OSF clause generalization rules.

The lattice of partially ordered type structures of (Aït-Kaci, 1983; Aït-Kaci and Sasaki, 2001) and the order-sorted equational setting of rewriting logic (Meseguer, 1998) differ in several aspects and are incomparable, *i.e.,* one is not subsumed into the other. The differences, explained below, are based on term representation, sort structure, and algebraic axioms. The order-sorted type structure is much simpler and typically finite, whereas the association of a type to each feature term makes the set of types infinite. In the much simpler order-sorted setting, only the subsort relations between basic sorts need to be *explicitly* considered, although *implicitly* each term with variables can be interpreted set-theoretically as the set of its substitution instances. Obviously, by an encoding of first order terms as feature-terms —the features simply being argument positions, *e.g.,* the term $f(t_1, .., t_n)$ if and only if the feature term $f(1 \Rightarrow t_1, .., , n \Rightarrow t_n)$— the order-sorted syntactic algorithm presented in (Alpuente et al., 2009b) could be seen as a special case of (Aït-Kaci, 1983). Conversely, feature types can also be expressed as algebraic types if we supply the missing constructors for attributes, which are called *implicit constructors* in (Smolka and Aït-Kaci, 1989), where this encoding was used to develop a frame-

work, based on equational constraint solving, where feature term unification and order-sorted term unification coexist. Thus, each term representations can be encoded into the other.

On the other hand, as already hinted at above, the sort structure is different in both approaches and, thus, the algorithm presented in (Aït-Kaci, 2007) is different from the one presented here. In (Aït-Kaci and Sasaki, 2001), least upper bounds (lubs) are canonically represented as disjunctive sets of maximal terms: if one wants to specify that an element is of sort A or B when no explicit type symbol is known as their lub, then this element is induced to be of type $A \vee B$. Instead, in an order-sorted setting (Goguen and Meseguer, 1992; Meseguer, 1998) the sort structure is much simpler, namely a (typically finite) *poset* as opposed to an infinite lattice. Yet, under the easily checkable assumption of *pre-regularity* (or E-pre-regularity for equational axioms E of associativity and/or commutativity and/or identity), each term (resp. each E-equivalence class of terms) has a *least sort* possible; see (Goguen and Meseguer, 1992), and (Clavel et al., 2007, 22.2.5). Furthermore, unlike the feature term case, there is no global assumption of a *top sort*, although each *connected component* in the poset of sorts can be conservatively extended with a top sort for that component; the so-called *kinds*, see (Clavel et al., 2007; Meseguer, 1998) and Section 2. This means that certain generalization problems are regarded as *incoherent* and have no solution. For example, there is no generalizer for the terms x:Bool and y:Nat, assuming that the connected components of sorts for numbers (where Nat is one of the sorts) and truth values (where Bool is another sort) are disjoint. Thus, the sort structure contains different assumptions in each approach.

Finally, even if the comma (conjunction) is handled in the OSF as an associative-commutative operator, the OSF does not support the definition of operators with combinations of algebraic properties such as commutativity, associativity and identity, while each operator in our order-sorted setting can have any desired combination of these algebraic properties.

2. Preliminaries

We follow the classical notation and terminology from (TeReSe, 2003) for term rewriting and from (Goguen and Meseguer, 1992; Meseguer, 1998) for order-sorted equational logic.

We assume an *order-sorted signature* $\Sigma = (S, F, \leq)$ that consists of a finite poset of sorts $(\mathsf{S}, \leq)$ and a family F of function symbols of the form $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$, with $\mathsf{s}_1, \ldots, \mathsf{s}_n, \mathsf{s} \in \mathsf{S}$. We furthermore assume a *kind-completed signature* such that: (i) each connected component in the poset ordering has a top sort, and for each $\mathsf{s} \in \mathsf{S}$ we denote by $[\mathsf{s}]$ the top sort in the connected component of s (*i.e.,* if s and s' are sorts in the same connected component, then $[\mathsf{s}] = [\mathsf{s}']$); and (ii) for each operator declaration $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$ in Σ, there is also a declaration $f : [\mathsf{s}_1] \times \ldots \times [\mathsf{s}_n] \to [\mathsf{s}]$ in Σ. A given term t in an order-sorted term algebra can have many different sorts. In particular, if $t \in T_\Sigma$ has sort s, then it also has sort s' for any $\mathsf{s}' \geq \mathsf{s}$; and because a function symbol

f can have different sort declaration $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$, a term $f(t_1, .., t_n)$ can have sorts that are not directly comparable (Goguen and Meseguer, 1992).

We assume a fixed S-sorted family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$ of pairwise disjoint variable sets (*i.e.,* $\mathcal{X}_\mathsf{s} \cap \mathcal{X}_{\mathsf{s}'} \neq \emptyset$), with each $\mathcal{X}_\mathsf{s}$ being countably infinite. We write the sort associated to a variable explicitly with a colon and the sort, *i.e.,* x:Nat. A *fresh* variable is a variable that appears nowhere else. The set $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ denotes all Σ-terms of sort s defined by $\mathcal{X}_\mathsf{s} \subseteq \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ and $f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ if $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s} \in \Sigma$ and $t_1 \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s}_1}, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s}_n}$. Furthermore, if $t \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ and $s \leq s'$, then $t \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s}'}$. For a term t, we write $Var(t)$ for the set of all variables in t. $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort s, *i.e.,* $t \in \mathcal{T}_{\Sigma,\mathsf{s}}$ if $Var(t) = \emptyset$. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding term algebras. We assume that $\mathcal{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ for every sort s.

We assume *pre-regularity* of the signature Σ: for each operator declaration $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$, and for the set S_f containing all sorts s' that appear in operator declarations of the form $f : \mathsf{s}'_1, \ldots, \mathsf{s}'_n \to \mathsf{s}'$ in Σ such that $\mathsf{s}_i \leq \mathsf{s}'_i$ for $1 \leq i \leq n$, then the set S_f has a least sort. Thanks to pre-regularity of Σ, each Σ-term t has a *unique least sort* which is denoted by $LS(t)$. The top sort in the connected component of $LS(t)$ is denoted by $[LS(t)]$. Since the poset $(\mathsf{S}, \leq)$ is finite and each connected component has a top sort, given any two sorts s and s' in the same connected component, the set of least upper bound sorts of s and s', although non necessarily a singleton set, always exists and is denoted by $LUBS(\mathsf{s}, \mathsf{s}')$.

Throughout this paper, we assume that Σ has no *ad-hoc operator overloading*, *i.e.,* any two operator declarations for the same symbol f with equal number of arguments, $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$ and $f : \mathsf{s}'_1 \times \ldots \times \mathsf{s}'_n \to \mathsf{s}'$, must necessarily have $[\mathsf{s}_1] = [\mathsf{s}'_1], \ldots, [\mathsf{s}_n] = [\mathsf{s}'_n], [\mathsf{s}] = [\mathsf{s}']$.

The set of positions of a term t, written $Pos(t)$, is represented as a sequence of natural numbers, *e.g.,* 1.2.1. The set of non-variable positions is written $Pos_\Sigma(t)$. The root position of a term is $\wedge$. The subterm of t at position p is $t|_p$ and $t[u]_p$ is the term obtained from t by replacing $t|_p$ by u. By $root(t)$ we denote the symbol occurring at the root position of t.

A *substitution* $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is a mapping from variables to terms which is almost everywhere equal to the identity except over a finite set of variables $\{x_1, \ldots, x_n\}$, written $Dom(\sigma) = \{x \in \mathcal{X} \mid x\sigma \neq x\}$. Substitutions are *sort–preserving*, *i.e.,* for any substitution σ, if $x \in \mathcal{X}_\mathsf{s}$, then $x\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$. We assume substitutions are idempotent, *i.e.,* $x\sigma = (x\sigma)\sigma$ for any variable x. The set of variables introduced by σ is $VRan(\sigma) = \bigcup\{ Var(x\sigma) \mid x\sigma \neq x\}$. The identity substitution is *id*. Substitutions are homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$. Substitutions are written in suffix notation (*i.e.,* $t\sigma$ instead of $\sigma(t)$), and, consequently, composition of substitutions must be read from left to right, formally denoted by juxtaposition, *i.e.,* $x(\sigma\sigma') = (x\sigma)\sigma'$ for any variable x. The restriction of σ to a set of variables V is $\sigma|_V$. We call a substitution σ a *renaming* if there is another substitution σ^{-1} such that $(\sigma\sigma^{-1})|_{Dom(\sigma)} = id$.

A Σ-*equation* is an unoriented pair $t \doteq t'$, where t and t' are Σ-terms for which there are sorts s, s' with $t \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$, $t' \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s}'}$, and s, s' are in the same connected component of the poset of sorts $(S, \leq)$. An *equational theory* (Σ, E)

is a set E of Σ-equations. An *equational theory* (Σ, E) over a kind-completed, pre-regular, and order-sorted signature $\Sigma = (S, F, \leq)$ is called kind-completed, pre-regular, and order-sorted equational theory. Given an equational theory (Σ, E), order-sorted equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, see (Goguen and Meseguer, 1992; Meseguer, 1998).

The *E-subsumption* preorder $\leq_E$ (simply $\leq$ when E is empty) holds between $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, denoted $t \leq_E t'$ (meaning that t is more general than t' modulo E), if there is a substitution σ such that $t\sigma =_E t'$; such a substitution σ is said to be an *E-matcher* for t' in t. The equivalence relation $\equiv_E$ (or $\equiv$ if E is empty) induced by $\leq_E$ is defined as $t \equiv_E t'$ if $t \leq_E t'$ and $t' \leq_E t$. The E-renaming equivalence $t \simeq_E t'$ (or $\simeq$ if E is empty), holds if there is a renaming substitution θ such that $t\theta =_E t'$. In general, the relations $=_E$, $\equiv_E$ and $\simeq_E$ do not coincide, actually $=_E \subseteq \simeq_E \subseteq \equiv_E$.

Example 1. *Consider terms $t = f(f(a, X), Y)$ and $t' = f(a, Z)$ where f is associative and commutative with identity symbol 0 (ACU), and a and b are two constants. We have that $t \equiv_{ACU} t'$, i.e., $t \leq_{ACU} t'$ and $t' \leq_{ACU} t$, since $f(f(a, X), Y)\sigma_1 =_{ACU} f(a, Z)$ with $\sigma_1 = \{X \mapsto 0, Y \mapsto Z\}$ and $f(a, Z)\sigma_2 =_{ACU} f(f(a, X), Y)$ with $\sigma_2 = \{Z \mapsto f(X, Y)\}$. However, $t \not\simeq_{ACU} t'$, since they are not equal up to ACU-renaming.*

3. Syntactic Least General Generalization

In order to better present our work, in this section we revisit untyped generalization (Huet, 1976; Plotkin, 1970; Reynolds, 1970) and formalize the lgg computation by means of a new inference system that will be useful in our subsequent extension of this algorithm to the order–sorted setting given in Section 4 and to the equational setting given in Section 5. Throughout this section, we assume unsorted terms, *i.e.,* $t \in \mathcal{T}_\Sigma(\mathcal{X})$, with an unsorted signature Σ. This can be understood as the special case of having only one sort in an order-sorted setting.

Most general unification of a (unifiable) set M of terms is given by the least upper bound (*most general instance, mgi*) of M under the standard instantiation quasi-ordering $\leq$ on terms given by the relation of being "more general". Formally,

$$\mathit{instances}(M) = \{t' \in \mathcal{T}_\Sigma(\mathcal{X}) \mid \forall t \in M,\ t \leq t'\}$$

and

$$\mathit{mgi}(M) = s \in \mathit{instances}(M) \text{ s.t. } \forall t' \in \mathit{instances}(M), s \leq t'.$$

Note that the most general instance is unique up to variable renaming. For instance, given the terms $f(f(x, x), b)$ and $f(f(b, b), y)$, the term $f(f(b, b), b)$ is the most general instance with the substitution $\{x \mapsto b, y \mapsto b\}$.

Least general generalization of M corresponds to the greatest lower bound, *i.e.,*

$$\mathit{generalizers}(M) = \{t' \in \mathcal{T}_\Sigma(\mathcal{X}) \mid \forall t \in M,\ t' \leq t\}$$

and

$$lgg(M) = s \in \mathit{generalizers}(M) \text{ s.t. } \forall t' \in \mathit{generalizers}(M), t' \leq s.$$

Note that the least general generalizer is unique up to variable renaming. For instance, given the terms $f(f(a,a),b)$ and $f(f(b,b),a)$, the term $f(f(x,x),y)$ is the least general generalizer with the two substitutions $\{x \mapsto a, y \mapsto b\}$ and $\{x \mapsto b, y \mapsto a\}$.

The non-deterministic generalization algorithm λ of Huet (Huet, 1976), also treated in detail in (Lassez et al., 1988), works as follows. Let Φ be any bijection between $\mathcal{T}_\Sigma(\mathcal{X}) \times \mathcal{T}_\Sigma(\mathcal{X})$ and a set of variables V. The recursive function λ on $\mathcal{T}_\Sigma(\mathcal{X}) \times \mathcal{T}_\Sigma(\mathcal{X})$ that computes the lgg of two terms is given by:

- $\lambda(f(s_1,\ldots,s_m), f(t_1,\ldots,t_m)) = f(\lambda(s_1,t_1),\ldots,\lambda(s_m,t_m))$, for $f \in \Sigma$
- $\lambda(s,t) = \Phi(s,t)$, otherwise.

Central to this algorithm is the global function Φ that is used to guarantee that the same disagreements are replaced by the same variable in both terms. different choices of Φ may result in different generalizers that are equivalent up to variable renaming.

In the following, we provide a novel set of inference rules for computing the (syntactic) least general generalization of two terms, first proposed in (Alpuente et al., 2009b), that uses a local store of already solved generalization subproblems. 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 effectively computing a complete and minimal set of least general generalizations in both, the order–sorted extension and the equational generalization algorithm developed in this article. A different formulation by means of inference rules is given in (Pottier, 1989), where the store is not explicit in the configurations but is implicitly kept within the constraint and substitution components, which is less intuitive and causes the accumulation of a lot of bindings for many variables with the same instantiations.

3.1. Novel inference rules for untyped, syntactic least general generalization

In our formulation, we represent a generalization problem between terms t and t' as a *constraint* $t \overset{x}{\triangleq} t'$, where x is a fresh variable that stands for a tentative generalizer of t and t'. By means of this representation, any generalizer w of t and t' is given by a suitable substitution θ such that $x\theta = w$. Note that, although a constraint $t \overset{x}{\triangleq} t'$ is commutative, the inference rules that are described in this paper do not admit that commutativity property, since it is very important to keep track of the origin of new generated generalization subproblems. However, we use the symbol $\wedge$ for a conjunction of constraints (i.e., $s_1 \overset{x_1}{\triangleq} t_1 \wedge \ldots \wedge s_n \overset{x_n}{\triangleq} t_n$) and this symbol is *associative* and *commutative* in the inference rules described in this paper.

We compute the least general generalization of t and t', written $lgg(t,t')$, by means of a transition system $(\mathit{Conf}, \rightarrow)$ (Plotkin, 2004) where Conf is a set

Decompose
$$\frac{f \in (\Sigma \cup \mathcal{X})}{\langle f(t_1,\ldots,t_n) \overset{x}{\triangleq} f(t'_1,\ldots,t'_n) \wedge C \mid S \mid \theta\rangle \rightarrow \langle t_1 \overset{x_1}{\triangleq} t'_1 \wedge \ldots \wedge t_n \overset{x_n}{\triangleq} t'_n \wedge C \mid S \mid \theta\sigma\rangle}$$
where $\sigma = \{x \mapsto f(x_1,\ldots,x_n)\}$, $x_1,\ldots,x_n$ are fresh variables, and $n \geq 0$

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

Recover
$$\frac{root(t) \neq root(t')}{\langle t \overset{x}{\triangleq} t' \wedge C \mid S \wedge t \overset{y}{\triangleq} t' \mid \theta\rangle \rightarrow \langle C \mid S \wedge t \overset{y}{\triangleq} t' \mid \theta\sigma\rangle}$$
where $\sigma = \{x \mapsto y\}$

Figure 2: Rules for least general generalization

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 the extra constraint component that we call the *store*.

Definition 1. *A configuration $\langle C \mid S \mid \theta\rangle$ consists of three components: (i) the* constraint component C*, 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*, (ii) the* store component S*, that records the* set of already solved constraints*, and (iii) 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 final (*i.e.,* terminal) configuration of the form $\langle \emptyset \mid S \mid \theta\rangle$, *i.e.,* a normal form w.r.t. the inference system, is reached. Then, the lgg of t and t' is given by $x\theta$. As we shall see, θ is unique up to renaming. Given a constraint $t \overset{x}{\triangleq} t'$, we call x an *index variable* or a variable at the *index position of the constraint*. Given a set C of constraints, each of the form $t \overset{x}{\triangleq} t'$ for some t, t', and x, we define the set of index variables as $Index(C) = \{y \in \mathcal{X} \mid \exists u \overset{y}{\triangleq} v \in C\}$.

The transition relation $\rightarrow$ is given by the smallest relation satisfying the rules in Figure 2. In this paper, variables of terms t and t' in a generalization problem $t \overset{x}{\triangleq} t'$ are considered as constants, and are never instantiated. The meaning of the rules is as follows.

- The **Decompose** rule is the syntactic decomposition generating new constraints to be solved.

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

Figure 3: Computation trace for (syntactic) generalization of terms $f(g(a),g(y),a)$ and $f(g(b),g(y),b)$

- The **Solve** rule checks that a constraint $t \overset{x}{\triangleq} t' \in C$, with $root(t) \neq root(t')$, is not already solved. If not already in the store S, then the solved constraint $t \overset{x}{\triangleq} t'$ is added to S.
- The **Recover** rule checks if a constraint $t \overset{x}{\triangleq} t' \in C$, with $root(t) \neq root(t')$, is already solved, *i.e.,* if there is already a constraint $t \overset{y}{\triangleq} t' \in S$ for the same pair of terms (t,t') with variable y. This is needed when the input terms of the generalization problem contain the same generalization subproblems more than once, *e.g.,* the lgg of $f(f(a,a),a)$ and $f(f(b,b),a)$ is $f(f(y,y),a)$.

Example 2. *Consider the terms $t = f(g(a),g(y),a)$ and $t' = f(g(b),g(y),b)$. In order to compute the least general generalizer of t and t', we apply the inference rules of Figure 2. The substitution component in the final configuration obtained by the lgg algorithm is $\theta = \{x \mapsto f(g(x_4),g(y),x_4), x_1 \mapsto g(x_4), x_2 \mapsto g(y), x_5 \mapsto y, x_3 \mapsto x_4\}$, hence the computed lgg is $x\theta = f(g(x_4),g(y),x_4)$. The execution trace is showed in Figure 3. Note that variable x_4 is repeated to ensure that the least general generalizer is obtained.*

3.2. Termination and Confluence of the untyped, syntactic least general generalization algorithm

Termination of the transition system $(Conf, \rightarrow)$ is straightforward.

Theorem 1 (Termination). *Every derivation stemming from an initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle$ using the inference rules of Figure 2 terminates in a final configuration of the form $\langle \emptyset \mid S \mid \theta\rangle$.*

PROOF. Let $|u|$ be the number of symbol occurrences in the syntactic object u. Since the minimum of $|t|$ and $|t'|$ is an upper bound to the number of times that the inference rule **Decompose** of Figure 2 can be applied, and the application of rules **Solve** and **Recover** strictly decreases the size $|C|$ of the C component of the lgg configurations at each step, then any derivation necessarily terminates and the constraint component in the final configuration is empty. □

Note that the inference rules of Figure 2 are non–deterministic (*i.e.,* they depend on the chosen constraint of the set C). However, in the following we show that they are confluent up to variable renaming (*i.e.,* the chosen transition is irrelevant for the computation of terminal configurations). This justifies the well-known fact that the least general generalizer of two terms is unique up to variable renaming (Lassez et al., 1988). In order to prove confluence of the calculus up to renaming, let us first prove an auxiliary result stating that only (independently) fresh variables y appear in the index positions of the constraints in C and S components of lgg configurations.

Lemma 1 (Uniqueness of Generalization Variables). *Let $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$ and $x \in \mathcal{X}$. For every derivation $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ stemming from the initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle$ using the inference rules of Figure 2, and for every $u \overset{y}{\triangleq} v \in C$ (similarly $u \overset{y}{\triangleq} v \in S$), the variable y does not appear in any other constraint in C or S, i.e., there are no other $u', v' \in \mathcal{T}_\Sigma(\mathcal{X})$ such that $u' \overset{y}{\triangleq} v' \in C$ or $u' \overset{y}{\triangleq} v' \in S$.*

PROOF. By induction on the length n of the sequence $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^n \langle C \mid S \mid \theta\rangle$. If $n = 0$, then the conclusion follows, since $C = t \overset{x}{\triangleq} t'$, $S = \emptyset$. If $n > 0$, then we split the derivation into $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^{n-1} \langle C' \mid S' \mid \theta'\rangle \rightarrow \langle C \mid S \mid \theta\rangle$ and we consider each inference rule of Figure 2 separately for the last step $\langle C' \mid S' \mid \theta'\rangle \rightarrow \langle C \mid S \mid \theta\rangle$ of the derivation sequence:

- **Decompose**. Here $C' = f(t_1, \ldots, t_n) \overset{x}{\triangleq} f(t'_1, \ldots, t'_n) \wedge C''$, $S = S'$, $C = t_1 \overset{x_1}{\triangleq} t'_1 \wedge \ldots \wedge t_n \overset{x_n}{\triangleq} t'_n \wedge C''$, and $\theta = \theta'\sigma$ where $\sigma = \{x \mapsto f(x_1, \ldots, x_n)\}$, $x_1, \ldots, x_n$ are fresh variables, and $n \geq 0$. Since the variables $x_1, \ldots, x_n$ are chosen to be fresh, they do not appear in either C'' or S'. The conclusion follows by the induction hypothesis, since any index variable in C'' does not appear more than once in C'' and does not appear at all in S'.

- **Solve**. Here $C' = t \overset{x}{\triangleq} t' \wedge C''$, $C = C''$, $S = S' \wedge t \overset{x}{\triangleq} t'$, $\theta = \theta'$, and the conclusion follows by induction hypothesis, since x does not appear in C'' or S'.

- **Recover**. Here $C' = t \overset{x}{\triangleq} t' \wedge C''$, $C = C''$, $S' = t \overset{y}{\triangleq} t' \wedge S''$, $S = S'$, $\theta = \theta'\sigma$, $\sigma = \{x \mapsto y\}$, and the conclusion follows by induction hypothesis, since both x and y do not appear in C'' or S''. □

Now we are ready to prove the confluence of the lgg computations.

Theorem 2 (Confluence). *The set of derivations stemming from any initial configuration* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle$ *using the inference rules of Figure 2 deliver a unique solution* $\langle \emptyset \mid S \mid \theta \rangle$ *up to renaming.*

Proof. Thanks to termination, the proof of confluence can be reduced to one of local confluence, i.e., to the confluence of any two single-step transitions from a configuration $\langle C \mid S \mid \theta \rangle$. Given a configuration $\langle t \overset{x}{\triangleq} t' \wedge C \mid S \mid \theta \rangle$, there is only one possible transition step applicable to $t \overset{x}{\triangleq} t'$ thanks to the non-overlapping inference rules of Figure 2. Thus, we consider below the case of having two constraints and their corresponding transitions.

Given any configuration $\langle t_1 \overset{y}{\triangleq} t_2 \wedge t'_1 \overset{y'}{\triangleq} t'_2 \wedge C \mid S \mid \theta \rangle$ stemming from the initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle$, we analyse each possible inference rule application to either $t_1 \overset{y}{\triangleq} t_2$ or $t'_1 \overset{y'}{\triangleq} t'_2$; we underline the relation $\rightarrow$ with the name of the inference rule used for transformation.

- If **Decompose** is applied to $t_1 \overset{y}{\triangleq} t_2$ (resp. $t'_1 \overset{y'}{\triangleq} t'_2$), then there is no interference between this rule application to $t_1 \overset{y}{\triangleq} t_2$ and any other rule application to $t'_1 \overset{y'}{\triangleq} t'_2$, since the **Decompose** rule is not recording information in the store S and is not retrieving information from the store S or the constraint C. Hence, given any two inference steps

  $$\langle t_1 \overset{y}{\triangleq} t_2 \wedge t'_1 \overset{y'}{\triangleq} t'_2 \wedge C \mid S \mid \theta \rangle \rightarrow_{\mathbf{Decompose}} \langle t'_1 \overset{y'}{\triangleq} t'_2 \wedge C_1 \mid S \mid \theta_1 \rangle$$

  and

  $$\langle t_1 \overset{y}{\triangleq} t_2 \wedge t'_1 \overset{y'}{\triangleq} t'_2 \wedge C \mid S \mid \theta \rangle \rightarrow_{\mathbf{Decompose/Solve}/Restore} \langle t_1 \overset{y}{\triangleq} t_2 \wedge C_2 \mid S_2 \mid \theta_2 \rangle,$$

  there are two configurations $\langle C_{12} \mid S_{12} \mid \theta_{12} \rangle$ and $\langle C_{21} \mid S_2 \mid \theta_{21} \rangle$ such that

  $$\langle t'_1 \overset{y'}{\triangleq} t'_2 \wedge C_1 \mid S \mid \theta_1 \rangle \rightarrow_{\mathbf{Decompose/Solve}/Restore} \langle C_{12} \mid S_{12} \mid \theta_{12} \rangle,$$

  $$\langle t_1 \overset{y}{\triangleq} t_2 \wedge C_2 \mid S_2 \mid \theta_2 \rangle \rightarrow_{\mathbf{Decompose}} \langle C_{21} \mid S_2 \mid \theta_{21} \rangle,$$

  and, by the uniqueness of index variables (Lemma 1), $\langle C_{12} \mid S_{12} \mid \theta_{12} \rangle$ and $\langle C_{21} \mid S_{21} \mid \theta_{21} \rangle$ are equal up to variable renaming. Thus, the conclusion follows.

- If **Recover** is applied to $t_1 \overset{y}{\triangleq} t_2$ (resp. $t'_1 \overset{y'}{\triangleq} t'_2$), we have the same conclusion as in the case of the **Decompose** rule, since the **Recover** rule is not recording information in the store S.

- If **Solve** is applied to $t_1 \overset{y}{\triangleq} t_2$ (resp. $t'_1 \overset{y'}{\triangleq} t'_2$), then we consider whether we have that $t_1 = t'_1$ and $t_2 = t'_2$ or not. If it is not true that $t_1 = t'_1$ and $t_2 = t'_2$, *i.e.,* $t_1 \neq t'_1$ or $t_2 \neq t'_2$, then the application of the **Solve** rule to $t_1 \overset{y}{\triangleq} t_2$ (resp. $t'_1 \overset{y'}{\triangleq} t'_2$) leads to the same conclusion that applying the **Decompose** and **Recover** rules. If $t_1 = t'_1$ and $t_2 = t'_2$, then the application of the inference rule **Solve** to $t_1 \overset{y}{\triangleq} t_2$ disables the application of the inference rule **Solve** to $t'_1 \overset{y'}{\triangleq} t'_2$ but enables the application of the inference rule **Recover** to $t'_1 \overset{y'}{\triangleq} t'_2$. That is, given the two inference steps

$$\langle t_1 \overset{y}{\triangleq} t_2 \wedge t_1 \overset{y'}{\triangleq} t_2 \wedge C \mid S \mid \theta\rangle \rightarrow_{\textbf{\textit{Solve}}} \langle t_1 \overset{y'}{\triangleq} t_2 \wedge C \mid S \wedge t_1 \overset{y}{\triangleq} t_2 \mid \theta\rangle$$

and

$$\langle t_1 \overset{y}{\triangleq} t_2 \wedge t_1 \overset{y'}{\triangleq} t_2 \wedge C \mid S \mid \theta\rangle \rightarrow_{\textbf{\textit{Solve}}} \langle t_1 \overset{y}{\triangleq} t_2 \wedge C \mid S \wedge t_1 \overset{y'}{\triangleq} t_2 \mid \theta\rangle,$$

we have that

$$\langle t_1 \overset{y'}{\triangleq} t_2 \wedge C \mid S \wedge t_1 \overset{y}{\triangleq} t_2 \mid \theta\rangle \rightarrow_{\textbf{\textit{Recover}}} \langle C \mid S \wedge t_1 \overset{y}{\triangleq} t_2 \mid \theta\{y' \mapsto y\}\rangle$$

and

$$\langle t_1 \overset{y}{\triangleq} t_2 \wedge C \mid S \wedge t_1 \overset{y'}{\triangleq} t_2 \mid \theta\rangle \rightarrow_{\textbf{\textit{Recover}}} \langle C \mid S \wedge t_1 \overset{y'}{\triangleq} t_2 \mid \theta\{y \mapsto y'\}\rangle.$$

Thus, $\langle C \mid S \wedge t_1 \overset{y}{\triangleq} t_2 \mid \theta\{y' \mapsto y\}\rangle$ and $\langle C \mid S \wedge t_1 \overset{y'}{\triangleq} t_2 \mid \theta\{y \mapsto y'\}\rangle$ are equal up to variable renaming, and the conclusion follows. □

3.3. Correctness and Completeness

Before proving correctness and completeness of the above inference rules, we introduce the auxiliary concepts of a conflict position and of conflict pairs, and three auxiliary lemmas. Also, we recall that, for a given constraint $t \overset{x}{\triangleq} t'$, the variable x is a valid generalizer of t and t', though generally not the least one.

The first lemma expresses that the range of the substitutions partially computed at any stage of a generalization derivation coincides with the set of the index variables of the configuration, except for the generalization variable x of the original generalization problem $t \overset{x}{\triangleq} t'$.

Lemma 2 (Range of Substitutions). *Given terms t and t' and a fresh variable x such that $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figure 2, then $Index(S \cup C) \subseteq VRan(\theta) \cup \{x\}$, and $VRan(\theta) = Var(x\theta)$.*

Proof. Immediate by construction (see Figure 2). □

The following lemma establishes an auxiliary property that is useful for defining the notion of a conflict pair of terms.

Lemma 3. *Given terms t and t' and a fresh variable x, then there is a sequence $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle u \stackrel{y}{\triangleq} v \wedge C \mid S \mid \theta\rangle$ using the inference rules of Figure 2 such that there is no variable z such that $u \stackrel{z}{\triangleq} v \in S$ if and only if there exists a position p of t and t' such that $t|_p = u$, $t'|_p = v$, and for all $p' < p$, $root(t|_{p'}) = root(t'|_{p'})$.*

Proof. Straightforward by successive applications of the inference rule **Decompose** of Figure 2. □

The notion of a conflict pair is the key idea for our generalization proof schema.

Definition 2 (Conflict Position/Pair). *Given terms t and t', a position $p \in \mathcal{P}os(t) \cap \mathcal{P}os(t')$ is called* a conflict position of t and t' *if $root(t|_p) \neq root(t'|_p)$ and for all $q < p$, $root(t|_q) = root(t'|_q)$. Given terms t and t', the pair (u, v) is called a* conflict pair of t and t' *if there exists at least one conflict position p of t and t' such that $u = t|_p$ and $v = t'|_p$.*

The following lemma expresses the appropriate connection between the constraints arising in a derivation and the conflict pairs of the initial configuration.

Lemma 4. *Given terms t and t' and a fresh variable x, then there is a sequence $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid u \stackrel{y}{\triangleq} v \wedge S \mid \theta\rangle$ using the inference rules of Figure 2 if and only if there exists a conflict position p of t and t' such that $t|_p = u$ and $t'|_p = v$.*

Proof. ($\Rightarrow$) Since $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid u \stackrel{y}{\triangleq} v \wedge S \mid \theta\rangle$, then there must be two configurations $\langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$, $\langle C_2 \mid u \stackrel{y}{\triangleq} v \wedge S_2 \mid \theta_2\rangle$ such that

$$\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle,$$

$$\langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle \rightarrow_{\mathbf{Solve}} \langle C_2 \mid u \stackrel{y}{\triangleq} v \wedge S_2 \mid \theta_2\rangle,$$

$$\langle C_2 \mid u \stackrel{y}{\triangleq} v \wedge S_2 \mid \theta_2\rangle \rightarrow^* \langle C \mid u \stackrel{y}{\triangleq} v \wedge S \mid \theta\rangle,$$

and, by application of the inference rule **Solve**, $root(u) \neq root(v)$. By using Lemma 3 for the derivation $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$, there exists a position p of t and t' such that $t|_p = u$ and $t'|_p = v$. Since $root(u) \neq root(v)$, p is a conflict position.

($\Leftarrow$) By Lemma 3, there is a configuration $\langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$ such that $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$, and $root(u) \neq root(v)$. Then, the inference rule **Solve** is applied, *i.e.*, $\langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle \rightarrow \langle C_1 \mid u \stackrel{y}{\triangleq} v \wedge S_1 \mid \theta_1\rangle$ and the constraint $u \stackrel{y}{\triangleq} v$ will be part of S in the final configuration $\langle \emptyset \mid S \mid \theta\rangle$. □

The following lemma establishes the link between the substitution component of a terminal configuration (simply called "computed substitution" from now on) and a proper generalizer.

Lemma 5. *Given terms t and t' and a fresh variable x,*

- *if $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figure 2, then $x\theta$ is a generalizer of t and t';*
- *if u is a generalizer of t and t', then, using the inference rules of Figure 2, $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ and $u \simeq x\theta$.*

Proof. By structural induction on the term $x\theta$ (resp. u). If $x\theta = x$ (resp. u) is a variable, then $\theta = id$ and the conclusion follows. If $x\theta = f(u_1, \ldots, u_k)$ (resp. $u = f(u_1, \ldots, u_k)$), then the **Decompose** inference rule is applied and we have that $t = f(t_1, \ldots, t_k)$ and $t' = f(t'_1, \ldots, t'_k)$. By induction hypothesis, u_i is a generalizer of t_i and t'_i, for $1 \leq i \leq k$. Now, if no variable is shared between two different u_i in $x\theta$ (resp. u), then the conclusion follows. Otherwise, for each variable z that is shared between two different terms u_i and u_j in $x\theta$ (resp. u), there is a constraint $w_1 \stackrel{z}{\triangleq} w_2$ in S and, by Lemma 4, there are conflict positions p_i in t_i and t'_i, and p_j in t_j and t'_j such that $t_i|_{p_i} = t_j|_{p_j} = w_1$ and $t'_i|_{p_i} = t'_j|_{p_j} = w_2$. Thus, the conclusion follows. □

Finally, correctness and completeness are proved as follows.

Theorem 3 (Correctness and Completeness). *Given terms t and t' and a fresh variable x, u is the lgg of t and t' if and only if there exists S and θ such that $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ using the inference rules of Figure 2 and $u \simeq x\theta$.*

Proof. We rely on the already known existence and uniqueness of the lgg of t and t' (Lassez et al., 1988) and reason by contradiction. Consider the normalizing derivation $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$. By Lemma 5, $x\theta$ is a generalizer of t and t'. If $x\theta$ is not the lgg of t and t' up to renaming, then there is a term u which is the lgg of t and t' and a substitution ρ which is not a variable renaming such that $x\theta\rho = u$. By Lemma 2, $VRan(\theta) = Var(x\theta)$, hence we can choose ρ with $Dom(\rho) = Var(x\theta)$. Now, since ρ is not a variable renaming, either:

1. there are variables $y, y' \in Var(x\theta)$ and a variable z such that $y\rho = y'\rho = z$, or
2. there is a variable $y \in Var(x\theta)$ and a non-variable term v such that $y\rho = v$.

In case (1), there are two conflict positions p, p' for t and t' such that $u|_p = z = u|_{p'}$ and $x\theta|_p = y$ and $x\theta|_{p'} = y'$. In particular, this means that $t|_p = t|_{p'}$ and $t'|_p = t'|_{p'}$. But this is impossible by Lemmas 4 and 2. In case (2), there is a position p such that $x\theta|_p = y$ and p is neither a conflict position of t and t' nor it is under a conflict position of t and t'. Since this is impossible by Lemmas 4 and 2, the claim is proved. □

Decompose
$$\frac{f \in (\Sigma \cup \mathcal{X}) \wedge f : [\mathsf{s}_1] \times \ldots \times [\mathsf{s}_n] \to [\mathsf{s}]}{\langle f(t_1, \ldots, t_n) \overset{x:[\mathsf{s}]}{\triangleq} f(t'_1, \ldots, t'_n) \wedge C \mid S \mid \theta \rangle \to \langle t_1 \overset{x_1:[\mathsf{s}_1]}{\triangleq} t'_1 \wedge \ldots \wedge t_n \overset{x_n:[\mathsf{s}_n]}{\triangleq} t'_n \wedge C \mid S \mid \theta\sigma \rangle}$$

where $\sigma = \{x{:}[\mathsf{s}] \mapsto f(x_1{:}[\mathsf{s}_1], \ldots, x_n{:}[\mathsf{s}_n])\}$, $x_1{:}[\mathsf{s}_1], \ldots, x_n{:}[\mathsf{s}_n]$ are fresh variables, and $n \geq 0$

Solve
$$\frac{root(t) \neq root(t') \wedge \mathsf{s}' \in LUBS(LS(t), LS(t')) \wedge \nexists y \nexists \mathsf{s}'' : t \overset{y:\mathsf{s}''}{\triangleq} t' \in S}{\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \wedge C \mid S \mid \theta \rangle \to \langle C \mid S \wedge t \overset{z:\mathsf{s}'}{\triangleq} t' \mid \theta\sigma \rangle}$$

where $\sigma = \{x{:}[\mathsf{s}] \mapsto z{:}\mathsf{s}'\}$ and $z{:}\mathsf{s}'$ is a fresh variable.

Recover
$$\frac{root(t) \neq root(t') \wedge \mathsf{s}'' \in LUBS(LS(t), LS(t'))}{\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \wedge C \mid S \wedge t \overset{y:\mathsf{s}'}{\triangleq} t' \mid \theta \rangle \to \langle C \mid S \wedge t \overset{y:\mathsf{s}'}{\triangleq} t' \mid \theta\sigma \rangle}$$

where $\sigma = \{x{:}[\mathsf{s}] \mapsto y{:}\mathsf{s}''\}$.

Figure 4: Rules for order–sorted least general generalization.

Let us mention that the generalization algorithm can also be used to compute (thanks to associativity and commutativity of symbol $\wedge$) the lgg of an arbitrary set of terms by successively computing the lgg of two elements of the set in the obvious way.

4. Order-sorted Least General Generalization

In this section, we generalize the unsorted generalization algorithm presented in Section 3 to the order-sorted setting.

We consider two terms t and t' having the same top sort, *i.e.,* $[LS(t)] = [LS(t')]$. Otherwise they are incomparable and no generalization exists. Starting from the initial configuration $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle$ where $[\mathsf{s}] = [LS(t)] = [LS(t')]$, configurations are transformed until a terminal configuration $\langle \emptyset \mid S \mid \theta \rangle$ is reached. In the order–sorted setting, the lgg, in general, is not unique. Each terminal configuration $\langle \emptyset \mid S \mid \theta \rangle$ provides an lgg of t and t' given by $(x{:}[\mathsf{s}])\theta$. A substitution δ is called *downgrading* if each binding in δ is of the form $x{:}\mathsf{s} \mapsto x'{:}\mathsf{s}'$, where x and x' are variable names and $\mathsf{s}' \leq \mathsf{s}$.

The transition relation $\to$ is given by the smallest relation satisfying the rules in Figure 4. The meaning of these rules is as follows.

- The **Decompose** rule is the syntactic decomposition generating new constraints to be solved. Fresh variables are initially assigned a top sort, which will be appropriately "downgraded" when necessary. Note that we

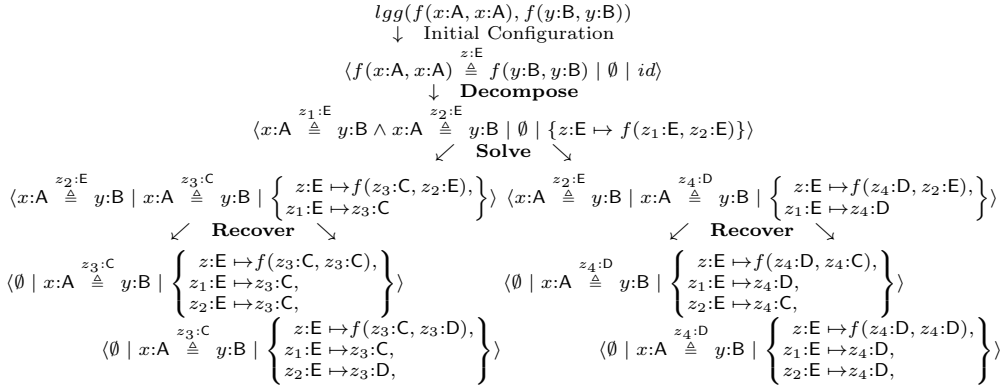

Figure 5: Computation for order–sorted generalization of terms $f(x{:}\mathsf{A}, x{:}\mathsf{A})$ and $f(y{:}\mathsf{B}, y{:}\mathsf{B})$

use only the top sort declaration of each symbol, instead of considering all the sort declarations for a symbol. The top sort declaration of a symbol is unique for kind-completed signatures and provides the most general sort for all the arguments of a symbol. It is safe to consider only the top sort declaration thanks to the pre-regularity of the signature.

- The **Solve** rule checks that a constraint $t \overset{x:[\mathsf{s}]}{\triangleq} t' \in C$, with $root(t) \neq root(t')$, is not already solved. Then, the solved constraint $t \overset{x:[\mathsf{s}]}{\triangleq} t'$ is added to the store S, and the substitution $\{x{:}[\mathsf{s}] \mapsto z{:}\mathsf{s}'\}$ is composed with the substitution part, where z is a fresh variable name with sort s' in the $LUBS$ of the least sorts of both terms.

- The **Recover** rule reuses a previously solved constraint. However, as an important difference with the corresponding unsorted rule of Figure 2, we cannot reuse both the variable identifier and the sort, and must consider all sorts in the $LUBS$ of the least sorts of both terms. This may seem unnecessary, since the $LUBS$ of the least sorts of both terms was considered in the previous application of the **Solve** rule, but it is necessary for completeness as Example 3 below shows.

Example 3. *Let $t = f(x{:}\mathsf{A}, x{:}\mathsf{A})$ and $t' = f(y{:}\mathsf{B}, y{:}\mathsf{B})$ be two terms where x and y are variables of sorts A and B, respectively, and assume the sort hierarchy that is shown in Figure 1. The top sort definition of f is $f : \mathsf{E} \to \mathsf{E}$. Starting from the initial configuration $\langle f(x{:}\mathsf{A}, x{:}\mathsf{A}) \overset{z:\mathsf{E}}{\triangleq} f(y{:}\mathsf{B}, y{:}\mathsf{B}) \mid \emptyset \mid id \rangle$, we apply the inference rules of Figure 4 and the substitutions obtained by the lgg algorithm are*

$$\theta_1 = \{z{:}\mathsf{E} \mapsto f(z_3{:}\mathsf{C}, z_3{:}\mathsf{C}), z_1{:}\mathsf{E} \mapsto z_3{:}\mathsf{C}, z_2{:}\mathsf{E} \mapsto z_3{:}\mathsf{C}\}$$

$$\theta_2 = \{z{:}\mathsf{E} \mapsto f(z_3{:}\mathsf{C}, z_3{:}\mathsf{D}), z_1{:}\mathsf{E} \mapsto z_3{:}\mathsf{C}, z_2{:}\mathsf{E} \mapsto z_3{:}\mathsf{D}\}$$
$$\theta_3 = \{z{:}\mathsf{E} \mapsto f(z_4{:}\mathsf{D}, z_4{:}\mathsf{C}), z_1{:}\mathsf{E} \mapsto z_4{:}\mathsf{D}, z_2{:}\mathsf{E} \mapsto z_4{:}\mathsf{C}\}$$
$$\theta_4 = \{z{:}\mathsf{E} \mapsto f(z_4{:}\mathsf{D}, z_4{:}\mathsf{D}), z_1{:}\mathsf{E} \mapsto z_4{:}\mathsf{D}, z_2{:}\mathsf{E} \mapsto z_4{:}\mathsf{D}\}$$

Note that all substitutions are incomparable, so that we have four posible lggs: $(z{:}\mathsf{E})\theta_1 = f(z_3{:}\mathsf{C}, z_3{:}\mathsf{C})$, $(z{:}\mathsf{E})\theta_2 = f(z_3{:}\mathsf{C}, z_3{:}\mathsf{D})$, $(z{:}\mathsf{E})\theta_3 = f(z_4{:}\mathsf{D}, z_4{:}\mathsf{C})$, *and* $(z{:}\mathsf{E})\theta_4 = f(z_4{:}\mathsf{D}, z_4{:}\mathsf{D})$. *The computation of these solutions is illustrated in Figure 5.*

Note that the inference rules **Recover** and **Solve** introduce an additional source of non-determinism (besides the choice of the next constraint to be processed) in our inference rules, in contrast to the syntactical rules of Figure 2. This extra non–determinism causes our rules to be non–confluent in general. However, this is essential for our algorithm to work, since different final configurations $\langle \emptyset \mid S_1 \mid \theta_1 \rangle, \ldots, \langle \emptyset \mid S_n \mid \theta_n \rangle$ correspond to different (least general) generalizers $x\theta_1, \ldots, x\theta_n$.

4.1. Termination, and Confluence for Single-Sorted Terms

Termination of the transition system $(\mathit{Conf}, \rightarrow)$ is straightforward.

Theorem 4 (Termination). *Every derivation stemming from an initial configuration* $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle$ *using the inference rules of Figure 4 where* $[\mathsf{s}] = [LS(t)] = [LS(t')]$ *terminates in a final configuration of the form* $\langle \emptyset \mid S \mid \theta \rangle$.

Proof. Similar to the proof of Theorem 1. □

The transition system $(\mathit{Conf}, \rightarrow)$ for order–sorted least general generalization given in Figure 4 is not confluent, as shown in Example 3. However, confluence can be recovered under appropriate conditions.

Definition 3 (Single-sort Constant). *Given an order-sorted signature* Σ*, we say that the constant* c *of sort* s *is* single-sorted *if* $\mathsf{s} = [\mathsf{s}]$.

Definition 4 (Single-sorted Variable). *A variable* $x{:}\mathsf{s}$ *is called* single-sorted *if* $\mathsf{s} = [\mathsf{s}]$.

Definition 5 (Single-sorted Term). *A term* t *is called* single-sorted *if every variable and every constant in* t *are single-sorted.*

In the following we assume a kind-completed, order-sorted pre-regular signature as described in Section 2.

Lemma 6. *Given a single-sorted term* t, $LS(t) = [LS(t)]$.

Proof. By structural induction on t. The cases when t is a variable or a constant are straightforward. If $t = f(t_1, \ldots, t_n)$, then by induction hypothesis, $LS(t_1) = [LS(t_1)], \ldots, LS(t_n) = [LS(t_n)]$, and given that $f : [\mathsf{s_1}] \times \cdots \times [\mathsf{s_n}] \rightarrow [\mathsf{s}]$, we have that $LS(t) = [LS(t)]$. □

Lemma 7. *Given two single-sorted terms t, t', $LUBS(LS(t), LS(t')) = LS(t) = LS(t') = [LS(t)] = [LS(t')]$.*

PROOF. By Lemma 6, $LS(t) = [LS(t)]$ and $LS(t') = [LS(t')]$ and, since $[LS(t)]$ is the top sort in the connected component, we conclude that $LUBS(LS(t), LS(t')) = \{LS(t)\} = \{LS(t')\}$. □

Theorem 5 (Confluence). *The set of derivations stemming from any initial configuration $\langle t \stackrel{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle$ using the inference rules of Figure 4, where t and t' are single-sorted terms and $[\mathsf{s}] = [LS(t)] = [LS(t')]$, deliver a unique solution $\langle \emptyset \mid S \mid \theta \rangle$ up to renaming.*

PROOF. Similar to the proof of Theorem 2, by taking into account that Lemma 7 ensures that there is no non-determinism involved in the application of the inference rules **Solve** and **Recover** of Figure 4. □

4.2. Order-sorted lgg computation by subsort specialization

Even if the set of least general generalizers of two terms is not generally a singleton, there is still a unique top-sorted generalizer that can just be specialized into the appropriate subsorts. This enables a different approach to computing order-sorted least general generalizers by just removing sorts (*i.e.,* upgrading variables to top sorts) in order to compute (unsorted) lgg's, and then obtaining the right subsorts by a suitable post-processing. In the following, we consider this naïve order-sorted generalization approach to provide, in Section 4.3 below, a proof of the correctness and completeness of the order-sorted least general generalization calculus of Figure 4. We also use this proof schema for the combined, order-sorted, equational least general generalization algorithm given in Section 6 below.

To simplify our notation, in the following we write $t[u]_{p_1,\ldots,p_n}$ instead of $((t[u]_{p_1}) \cdots)[u]_{p_n}$. The notion of conflict pair of Definition 2 can be extended to the order-sorted case in the obvious way, since two variables of different sorts having the same name, *e.g.,* $x{:}\mathsf{s_1}$ and $x{:}\mathsf{s_2}$, are considered to be different.

Definition 6 (Top-sorted Generalizer). *Given terms t and t' such that $[LS(t)] = [LS(t')]$, let $(u_1, v_1), \ldots, (u_k, v_k)$ be the conflict pairs of t and t', and for each such conflict pair (u_i, v_i), let $p^i_1, \ldots, p^i_{n_i}$, $1 \leq i \leq k$, be the corresponding conflict positions (i.e., $t|_{p^i_j} = u_i$ and $t'|_{p^i_j} = v_i$ for $1 \leq j \leq n_i$), and let $[\mathsf{s_i}] = [LS(u_i)] = [LS(v_i)]$. The* top-sorted generalization *of t and t' is defined by*

$$tsg(t, t') = t[x^1_1{:}[\mathsf{s_1}], \ldots, x^1_{n_1}{:}[\mathsf{s_1}]]_{p^1_1,\ldots,p^1_{n_1}} \cdots [x^k_1{:}[\mathsf{s_k}], \ldots, x^k_{n_k}{:}[\mathsf{s_k}]]_{p^k_1,\ldots,p^k_{n_k}}$$

where $x^1_1{:}[\mathsf{s_1}], \ldots, x^1_{n_1}{:}[\mathsf{s_1}], \ldots, x^k_1{:}[\mathsf{s_k}], \ldots, x^k_{n_k}{:}[\mathsf{s_k}]$ are fresh variables.

Example 4. *Let us consider the terms $t = f(x{:}\mathsf{A}, x{:}\mathsf{A})$ and $t' = f(y{:}\mathsf{B}, y{:}\mathsf{B})$ of Example 3. We have that $tsg(t, t') = f(z^1_1{:}\mathsf{E}, z^1_2{:}\mathsf{E})$, since there is only one conflict pair $(x{:}\mathsf{A}, y{:}\mathsf{B})$ with two confict positions (i.e., $k = 1$, $n_1 = 2$, $p^1_1 = 1$, and $p^1_2 = 2$) and $[\mathsf{A}] = [\mathsf{B}] = \mathsf{E}$.*

Once the unique top-sorted lgg is generated, the order-sorted lgg's are obtained by subsort specialization.

Definition 7 (Sort-specialized Generalizers). *Given terms t and t' such that $[LS(t)] = [LS(t')]$, let $(u_1, v_1), \ldots, (u_k, v_k)$ be the conflict pairs of t and t', and for each such conflict pair (u_i, v_i), let $p_1^i, \ldots, p_{n_i}^i$, $1 \leq i \leq k$, be the corresponding conflict positions (i.e., $t|_{p_j^i} = u_i$ and $t'|_{p_j^i} = v_i$ for $1 \leq j \leq n_i$), let $[\mathsf{s_i}] = [LS(u_i)] = [LS(v_i)]$, and let $x_1^1{:}[\mathsf{s_1}], \ldots, x_{n_1}^1{:}[\mathsf{s_1}], \ldots, x_1^k{:}[\mathsf{s_k}], \ldots, x_{n_k}^k{:}[\mathsf{s_k}]$ be the variable identifiers used in Definition 6. We define*

$$\begin{aligned} \textit{sort-down-subs}(t, t') = \{\rho \mid\ & \textit{Dom}(\rho) = \{x_1^1{:}[\mathsf{s_1}], \ldots, x_{n_1}^1{:}[\mathsf{s_1}], \ldots, x_1^k{:}[\mathsf{s_k}], \ldots, x_{n_k}^k{:}[\mathsf{s_k}]\} \wedge \\ & \forall 1 \leq i \leq k, \forall 1 \leq j \leq n_i : \\ & (x_j^i{:}[\mathsf{s_i}])\rho = x_i{:}\mathsf{s_i'} \wedge \mathsf{s_i'} \in \mathit{LUBS}(\mathit{LS}(u_i), \mathit{LS}(v_i))\} \end{aligned}$$

where all the $x_i{:}\mathsf{s_i'}$ are fresh variables.

The set of sort-specialized generalizers of t and t' is defined as $ssg(t, t') = \{tsg(t, t')\rho \mid \rho \in \textit{sort-down-subs}(t, t')\}$.

Example 5. *Continuing Example 4, we have that $\textit{sort-down-subs}(t, t') = \{\{z_1^1{:}\mathsf{E} \mapsto z_3{:}\mathsf{C}, z_2^1{:}\mathsf{E} \mapsto z_3{:}\mathsf{C}\}, \{z_1^1{:}\mathsf{E} \mapsto z_3{:}\mathsf{C}, z_2^1{:}\mathsf{E} \mapsto z_3{:}\mathsf{D}\}, \{z_1^1{:}\mathsf{E} \mapsto z_4{:}\mathsf{D}, z_2^1{:}\mathsf{E} \mapsto z_4{:}\mathsf{C}\}, \{z_1^1{:}\mathsf{E} \mapsto z_4{:}\mathsf{D}, z_2^1{:}\mathsf{E} \mapsto z_4{:}\mathsf{D}\}\}$ and so $ssg(t, t') = \{f(z_3{:}\mathsf{C}, z_3{:}\mathsf{C}), f(z_3{:}\mathsf{C}, z_3{:}\mathsf{D}), f(z_4{:}\mathsf{D}, z_4{:}\mathsf{C}), f(z_4{:}\mathsf{D}, z_4{:}\mathsf{D})\}$.*

The following result establishes that sort-specialized generalization and the order-sorted least general generalization do coincide.

Theorem 6. *Given terms t and t' such that $[LS(t)] = [LS(t')]$, it holds that 1) $tsg(t, t')$ is a generalizer of t and t', and 2) $ssg(t, t')$ provides a minimal and complete set of order-sorted lggs.*

Proof. It is immediate that $tsg(t, t')$ is a generalizer of t and t' since, for each occurrence of a conflict pair (s, s'), the term $tsg(t, t')$ contains a different variable at the corresponding conflict positions of t and t' which has the top sort associated to s and s'.

We prove that $sgg(t, t')$ provides a minimal and complete set of order-sorted lggs by contradiction. First, let us prove that it is complete by assuming that there is a generalizer u of t and t' s.t. there is no $u' \in ssg(t, t')$ with $u \leq u'$. Since $tsg(t, t')$ is a generalisation of t and t', we have that either $u \leq tsg(t, t')$ or $tsg(t, t') \leq u$. If $tsg(t, t') \leq u$, then at least one of the variables $x_j^i{:}[\mathsf{s_i}]$ of $tsg(t, t')$ corresponding to a conflict pair of t and t' must have been instantiated with a variable $z{:}\mathsf{s}$ such that $\mathsf{s} \leq [\mathsf{s_i}]$. Note that variable $x_j^i{:}[\mathsf{s_i}]$ cannot be instantiated with any symbol, since there is a conflict pair at that position. But then, since $ssg(t, t')$ considers all the sorts in the LUBS of the sorts of t and t', then there must be a term $u' \in ssg(t, t')$ such that $u \leq u'$. On the other hand, if $u \leq tsg(t, t')$, there must be a term $u' \in ssg(t, t')$ such that $u \leq u'$. Thus, the conclusion follows.

Second, let us prove that it is minimal by assuming that there are two generalizers u, u' of t and t' s.t. $u \in ssg(t, t')$, $u' \in ssg(t, t')$, $u \leq u'$ and $u' \leq u$. If $u \simeq u'$, then this is not possible, since the set $ssg(t, t')$ does not produce two generalisations that are equal up to renaming. If $u \not\simeq u'$, then it is not possible for the order-sorted case to have that $u \leq u'$ and $u' \leq u$. Thus, the conclusion follows. □

4.3. Correctness and Completeness of the order-sorted lgg calculus

Before proving the correctness and completeness of the order-sorted lgg calculus given in Figure 4, we provide some auxiliary notions and lemmas.

The first lemma relates the constraints that arise in an order-sorted, least general generalization derivation with positions of the input terms t and t'.

Lemma 8. *Given terms t and t' such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, and a fresh variable $x{:}[\mathsf{s}]$, then there is a sequence $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle u \overset{y:[\mathsf{s}']}{\triangleq} v \wedge C \mid S \mid \theta\rangle$ using the inference rules of Figure 4 such that there is no variable z such that $u \overset{z}{\triangleq} v \in S$ if and only if there exists a position p of t and t' such that $t|_p = u$ and $t'|_p = v$, and $[\mathsf{s}'] = [LS(u)] = [LS(v)]$.*

Proof. Straightforward by successive applications of the **Decompose** inference rule of Figure 4. □

The following lemma links the constraints already solved (and thus saved in the store) with conflict positions of the input terms t and t'

Lemma 9. *Given terms t and t' such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, and a fresh variable $x{:}[\mathsf{s}]$ such that $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ using the inference rules of Figure 4, the constraint $u \overset{y:\mathsf{s}'}{\triangleq} v$ belongs to S if and only if there exists a conflict pair (u, v) of t and t' such that $\mathsf{s}' \in LUBS(LS(u), LS(v))$.*

Proof. ($\Rightarrow$) If $u \overset{y:\mathsf{s}'}{\triangleq} v \in S$, then there must be a sort s'' and two configurations $\langle u \overset{y:[\mathsf{s}'']}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$, $\langle C_2 \mid u \overset{y:\mathsf{s}'}{\triangleq} v \wedge S_2 \mid \theta_2\rangle$ such that

$$\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle u \overset{y:[\mathsf{s}'']}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle \rightarrow \langle C_2 \mid u \overset{y:\mathsf{s}'}{\triangleq} v \wedge S_2 \mid \theta_2\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle,$$

where $\mathsf{s}' \leq [\mathsf{s}'']$, and, by application of the inference rule **Solve**, $root(u) \neq root(v)$. By using Lemma 8 for the derivation $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle u \overset{y:[\mathsf{s}'']}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$, there exists a position p of t and t' such that $t|_p = u$, $t'|_p = v$, and $[\mathsf{s}''] = [LS(u)] = [LS(v)]$. Since $root(u) \neq root(v)$, p is a conflict position. Then, by application of the inference rule **Solve**, we have that $\mathsf{s}' \in LUBS(LS(u), LS(v))$.

($\Leftarrow$) By Lemma 8, there exist a sort $[\mathsf{s}'']$ and a configuration $\langle u \overset{y:[\mathsf{s}'']}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$ such that $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle u \overset{y:[\mathsf{s}'']}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$, and $root(u) \neq root(v)$. Then, the inference rule **Solve** is applied, *i.e.,* $\langle u \overset{y:[\mathsf{s}'']}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle \rightarrow_{\mathbf{Solve}} \langle C_1 \mid u \overset{y:\mathsf{s}'}{\triangleq} v \wedge S_1 \mid \theta_1\rangle$, and $\mathsf{s}' \in LUBS(LS(u), LS(v))$. Hence, the constraint $u \overset{y:\mathsf{s}'}{\triangleq} v$ will be part of S in the final configuration $\langle \emptyset \mid S \mid \theta\rangle$. □

Lemma 10. *Given terms t and t' such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, for all S and θ such that $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ using the inference rules of Figure 4, there exists a downgrading substitution δ such that $tsg(t,t')\delta = (x{:}[\mathsf{s}])\theta$.*

Proof. By successive applications of the **Decompose** inference rule of Figure 4 we obtain that the term $tsg(t,t')$ is reproduced completely by $(x{:}[\mathsf{s}])\theta$, except for the variables $x^i_j{:}[\mathsf{s_i}]$. □

Theorem 7 (Correctness and Completeness). *Given terms t and t' such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, and a fresh variable $x{:}[\mathsf{s}]$, it holds that u is an order-sorted lgg of t and t' if and only if there exist S and θ such that $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ using the inference rules of Figure 4 and $u \simeq (x{:}[\mathsf{s}])\theta$.*

Proof. We reason by contradiction.

($\Rightarrow$) Let us consider a store S and substitution θ such that there is no term u and renaming ρ with $u\rho = (x{:}[\mathsf{s}])\theta$. By Theorem 6, $tsg(t,t') \leq u$ with a downgrading substitution δ_u, *i.e.,* $tsg(t,t')\delta_u = u$. By Lemma 10, $tsg(t,t') \leq (x{:}[\mathsf{s}])\theta$ with a downgrading substitution δ, *i.e.,* $tsg(t,t')\delta = (x{:}[\mathsf{s}])\theta$.

Since $(x{:}[\mathsf{s}])\theta$ and u are not renamed variants and both terms are sort-specializations of $tsg(t,t')$, there must be one binding $x{:}[\mathsf{s}] \mapsto x'{:}\mathsf{s}'$ in δ and one binding $x{:}[\mathsf{s}] \mapsto x''{:}\mathsf{s}''$ in δ_u s.t. either $\mathsf{s}' < \mathsf{s}''$, $\mathsf{s}'' < \mathsf{s}'$, or $\mathsf{s}' \neq \mathsf{s}''$ (where $[\mathsf{s}]{=}[\mathsf{s}']{=}[\mathsf{s}'']$). But all three possibilities are impossible by construction, since $\mathsf{s}' < \mathsf{s}''$ contradicts the fact that u is a lgg, $\mathsf{s}'' < \mathsf{s}'$ contradicts Lemma 9, and $\mathsf{s}' \neq \mathsf{s}''$ contradicts both that u is a lgg of t and t' and Lemma 9.

($\Leftarrow$) This direction can be proven similarly. □

5. 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 any (syntactic) least general generalization. Similarly to the dual case of E-unification, we have to talk about a *set* of least general E-generalizers, see, *e.g.,* (Baader, 1991), as shown in the following example.

Example 6. *Consider terms $t = f(f(a,a),b)$ and $s = f(f(b,b),a)$ where f is associative and commutative (AC), and a and b are two constants. Terms*

$u = f(f(x,x),y)$ and $u' = f(f(x,a),b)$ are generalizers 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. Furthermore, both u and u' are most specific generalizers of t and s.

A set M_1 of terms is said to be a *complete E-generalization* of another set M_2 of terms if for each term $t_2 \in M_2$, there is a term $t_1 \in M_1$ such that $t_1 \leq_E t_2$. A set M of terms is said to be *E-minimal* if for any two terms $t_1, t_2 \in M$ such that $t_1 \leq_E t_2$, then $t_1 = t_2$. For a set M of terms, we define the set of most specific generalizers of M modulo E as a set of *maximal lower bounds* of M under $\leq_E$, *i.e.,* $lgg_E(M)$ is a E-minimal, complete E-generalization of M.

As for the minimality, having several incomparable equational lggs is much tricky because the relation $\equiv_E$ induced by $\leq_E$ is in general different to the renaming relation $\simeq_E$ (except when $E = \emptyset$), as shown in the following example.

Example 7. *Consider terms $t = f(f(a,b),c)$ and $s = f(f(a,b),d)$ where f is associative and commutative with identity symbol 0 (ACU) and a, b, c and d are constants. The terms $f(a,f(b,x))$, $f(a,f(b,f(x,y)))$, $f(a,f(b,f(x,f(y,z))))$, ... are generalizers of t and s, and all of them are comparable w.r.t. $\leq_{ACU}$ (indeed, they are in the same equivalence class w.r.t the relation $\equiv_{ACU}$ even if they are not ACU-renamings of each other); hence we have to choose just one of them in order to obtain a minimal and complete set of ACU least general generalizers of t and s, e.g., $lgg_{ACU}(t,s) = \{f(a,f(b,x))\}$.*

In the following, we first adress the problem of computing a complete set of equational lggs and then we distill a minimal set by filtering away the redundant generalizers.

5.1. Recursively enumerating the least general generalizers modulo E

Given a finite set of equations E, and two terms t and s, we can always recursively enumerate the set that is by construction a complete set of generalizers of t and s. For this, we only need to recursively enumerate all pairs of terms (u,u') with $t =_E u$ and $s =_E u'$ and compute $lgg(u,u')$.

Definition 8. *Let t and s be terms and let E be an equational theory. A complete set of generalizers of t and s modulo E, denoted by $gen_E(t,t')$, is defined as follows:*

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

Let us prove that the set $\mathrm{gen}_E(t,t')$ is a complete set of E-lggs.

Lemma 11. *Given terms t and t' in an equational theory E, if u is a least general generalizer modulo E of t and t', then there exists $u' \in gen_E(t,t')$ such that $u' \simeq_E u$.*

PROOF. By contradiction. Let u be a lgg of t and t' modulo E and assume that there is no $u' \in gen_E(t,t')$ such that $u' \equiv_E u$. Since u is a lgg modulo E of t and t', then $u \leq_E t$ and $u \leq_E t'$. That is, there exist substitutions σ_t and $\sigma_{t'}$ such that $u\sigma_t =_E t$ and $u\sigma_{t'} =_E t'$. But, since u is a lgg of t and t' modulo E, there is no other term v such that $u \leq_E v$, $v \leq_E t$ and $v \leq_E t'$, which implies that, for each binding $X \mapsto u_t$ in σ_t and each binging $X \mapsto u_{t'}$ in $\sigma_{t'}$, the lgg modulo E of u_t and $u_{t'}$ is a variable. Then, u is a syntactic lgg of $u\sigma_t$ and $u\sigma'_t$ up to variable renaming and, by definition, there is $u' \in gen_E(t,t')$ such that $u' \simeq_E u$, which contradicts the assumption. □

Of course, the set $gen_E(t,t')$ may easily be infinite (*e.g.,* when including identity axioms). 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, minimal, and complete set of generalizers of t and s.

In any case, for any finite set of equations E, we can always mathematically characterize a *minimal complete set* of E-generalizers, namely the set $lgg_E(t,s)$ defined as follows. Roughly speaking, the minimal and complete set $lgg_E(t,s)$ is just one of the minimal sets than can be obtained from the complete (generally non-minimal) set $gen_E(t,s)$ by filtering only the maximal elements of the set with regard to the ordering $\leq_E$, as also noted in (Pottier, 1989).

Definition 9. *All the possible sets of least general generalizers of t and s modulo E are defined as follows:*

$$LGG_E(t,s) = \{G \subseteq gen_E(t,t') \mid G \text{ is a complete } E\text{-generalization of } gen_E(t,t') \text{ and } E\text{-minimal}\}.$$

Then, a set $lgg_E(t,s)$ of least general generalizers of t and s modulo E is defined by choosing G such that $G \in LGG_E(t,s)$.

Now, the minimality and completeness result for $lgg_E(t,t')$ follows straightforwardly.

Theorem 8. *Given terms t and t' in an equational theory E, $lgg_E(t,t')$ is a minimal, correct, and complete set of lggs modulo E of t and t'.*

PROOF. Lemma 11 ensures that $gen_E(t,t')$ is a complete set of lggs. Minimality and completeness of the set $lgg_E(t,t')$ is ensured by definition. □

Note that it may be the case that the subsumption relation $t \leq_E t'$ is *undecidable*, so that the above set of least general generalizers, although definable at the mathematical level, might not be effectively computable. 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 $t \leq_E t'$ is precisely the E-matching relation.

In summary, when E is finite and satisfies conditions (i) and (ii), the above definitions give us a feasible procedure to compute a finite, minimal, and complete set of least general generalizers $lgg_E(t, s)$. However, this procedure is horribly inefficient, because the cardinality of the E-equivalence classes can be exponential in the size of their elements, as in the case of associative-commutative theories (Pottier, 1989): for instance, if f is AC, then the class E for $f(a_1, f(a_2, ..., f(a_{n-1}, a_n)...))$ has $(2n-2)!!/((n-1)!$ elements. 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, when we add identiy axioms, E-equivalence classes become infinite, so that the above approach no longer gives us a lgg algorithm modulo E.

In the following sections, we do provide a modular, terminating, sound, and complete algorithm for equational theories containing different axioms such as associativity, commutativity, and identify (and their combinations). Our modular algorithm defined below does not provide, a priori, a minimal set of least general generalizers, so that it must be filtered out to obtain a set of least general generalizers, *i.e.,* one of the possible available sets $lgg_E(t, t')$. That is: first a complete set of E-generalizers is computed by the inference rules given below, and then 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 that provides the relation $\leq_E$.

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, *i.e.,* $ax(f) = \emptyset$. Note that, technically, variables of the original terms are handled in our inference rules as constants, thus without any attribute, *i.e.,* for any variable $x \in X$, we consider $ax(x) = \emptyset$.

Let us provide our inference rules for equational generalization in a stepwise manner. First, $ax(f) = \emptyset$ in Section 5.2, then, $ax(f) = \{C_f\}$ in Section 5.3, then, $ax(f) = \{A_f\}$ in Section 5.4, then, $ax(f) = \{A_f, C_f\}$ in Section 5.5, and finally, $U_f \in ax(f)$ in Section 5.6. In each section, proofs of correctness and completeness are very similar to the ones in Section 3.3 and, thus, we define two key notions, *pair of subterms* and *conflict pair*, for each equational property (*i.e., commutative pair of subterms*, *associative pair of subterms*, *associative and commutative pair of subterms*, and *identity pair of subterms* plus *commutative conflict pairs*, *associative conflict pairs*, *associative-commutative conflict pairs*, and *identity conflict pairs*) which are the basis for our overall proof scheme. For readability, we have provided complete proofs, even if they are in several aspects similar and differ mainly in the different conflict pair notions, which make it impossible to structure the proof in a parametric way.

5.2. Basic inference rules for least general E–generalization

Let us start with a set of basic rules in Figure 6 that are the equational version of the syntactic generalization rules of Section 3. The $\mathbf{Decompose}_E$ rule applies to function symbols obeying no axioms, $ax(f) = \emptyset$. Specific rules

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

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

$$\textbf{Solve}_E \quad \frac{f = root(t) \wedge g = root(t') \wedge f \neq g \wedge U_f \notin ax(f) \wedge U_g \notin ax(g) \wedge \nexists y : t \overset{y}{\triangleq} t' \in^E S}{\langle t \overset{x}{\triangleq} t' \wedge C \mid S \mid \theta\rangle \rightarrow \langle C \mid S \wedge t \overset{x}{\triangleq} t' \mid \theta\rangle}$$

$$\textbf{Recover}_E \quad \frac{root(t) \neq root(t') \wedge \exists y : t \overset{y}{\triangleq} t' \in^E S}{\langle t \overset{x}{\triangleq} t' \wedge C \mid S \mid \theta\rangle \rightarrow \langle C \mid S \mid \theta\sigma\rangle}$$

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

Figure 6: Basic inference rules for least general E–generalization

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 $\textbf{Solve}_E$ and $\textbf{Recover}_E$, the main difference w.r.t. the corresponding syntactic generalization rules given in Section 3 is in the fact that, for asking the store, we consider the constraints modulo E: in the rules below, we write $(t \overset{y}{\triangleq} t') \in^E S$ to express that there exists $u \overset{y}{\triangleq} u' \in S$ such that $t =_E u$ and $t' =_E u'$.

Finally, regarding the $\textbf{Solve}_E$ rule, 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 $\textbf{Expand}_U$ is given in Section 5.6 that gives us the opportunity to solve a constraint (conflict pair) $f(t_1,t_2) \overset{x}{\triangleq} s$, such that $root(s) \neq f$, with a generalizer $f(y,z)$ more specific than x, by first introducing the constraint $f(t_1,t_2) \overset{x}{\triangleq} f(s,e)$.

Termination, correctness and completeness of the basic algorithm are straightforward by reasoning similarly to the syntactic case of Section 3.

Theorem 9 (Termination). *Given an equational theory (Σ, E), Σ-terms t and t' such that every symbol in t and t' is free, and a fresh variable x, every derivation stemming from an initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle$ using the inference rules of Figure 6 terminates in a final configuration of the form $\langle \emptyset \mid S \mid \theta\rangle$.*

Proof. It follows directly from Theorem 1. □

Theorem 10 (Correctness and Completeness). *Given an equational theory (Σ, E), Σ-terms t and t' such that every symbol in t and t' is free, and a fresh variable x, then u is the lgg of t and t' if and only if there exist S and θ such that $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ using the inference rules of Figure 6, and $u \simeq x\theta$.*

Decompose$_C$

$$\frac{C_f \in ax(f) \wedge A_f \notin ax(f) \wedge i \in \{1,2\}}{\langle f(t_1,t_2) \overset{x}{\triangleq} f(t'_1,t'_2) \wedge C \mid S \mid \theta\rangle \to \langle t_1 \overset{x_1}{\triangleq} t'_i \wedge t_2 \overset{x_2}{\triangleq} t'_{(i \bmod 2)+1} \wedge C \mid S \mid \theta\sigma\rangle}$$

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

Figure 7: Decomposition rule for a commutative function symbol f

Proof. It follows directly from Theorem 3. □

Note that the basic inference rules of Figure 6 are confluent when $E = \emptyset$, according to Theorem 2, but the inference system of Section 6 that combines free, commutative, associative, and associative-commutative operators with and without an identity element is not generally confluent, and different final configurations $\langle \emptyset \mid S_1 \mid \theta_1\rangle, \ldots, \langle \emptyset \mid S_n \mid \theta_n\rangle$ correspond to different (least general) generalizers $x\theta_1, \ldots, x\theta_n$.

5.3. 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 7, for dealing with commutative function symbols. This inference rule replaces the syntactic decomposition inference rule for the case of a binary commutative symbol f, *i.e.,* the two possible rearrangements of the terms $f(t_1,t_2)$ and $f(t'_1,t'_2)$ are considered. Just notice that this rule is (don't know) non-deterministic, hence all four combinations must be explored.

Example 8. *Let $t = f(a,b)$ and $s = f(b,a)$ be two terms where f is commutative, i.e., $ax(f) = \{C_f\}$. Starting from the initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle$, 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 $x\theta = f(b,a)$. There are other three derivations, one returning $f(a,b)$ as an lgg, which is equivalent to the previous lgg, and two other ones returning $f(x,y)$ and $f(x',y')$ which are more general than the previous lgg.*

Termination is straightforward.

Theorem 11 (Termination). *Given an equational theory (Σ, E), Σ-terms t and t' such that every symbol in t and t' is either free or commutative, and a fresh variable x, every derivation stemming from an initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle$ using the inference rules of Figures 6 and 7 terminates in a final configuration of the form $\langle \emptyset \mid S \mid \theta\rangle$.*

Proof. Similar to the proof of Theorem 1 by considering the two possible rearrangements of each term. □

In order to prove correctness and completeness of the lgg calculus modulo C, similarly to Definition 2 we introduce the auxiliary concepts of *commutative pair of subterms* and *commutative conflict pair*, and prove some useful results for this case.

First, an auxiliary result is given stating that only (independently) fresh variables y appear in the index positions of the constraints in C and S components of lgg configurations.

Lemma 12 (Uniqueness of Generalization Variables). *Lemma 1 holds for* $t \overset{x}{\triangleq} t'$ *when the symbols in* t, t' *are free or commutative, for the inference rules of Figures 6 and 7.*

The following lemma expresses that the range of the substitutions partially computed at any stage of a generalization derivation coincides with the set of the index variables of the configuration, except for the generalization variable x of the original generalization problem $t \overset{x}{\triangleq} t'$.

Lemma 13 (Range of Substitutions). *Lemma 2 holds for* $t \overset{x}{\triangleq} t'$ *when the symbols in* t, t' *are free or commutative, for the inference rules of Figures 6 and 7.*

The following definition establishes an auxiliary property that is useful for defining the notion of a commutative conflict pair of terms. The depth of a position is defined as $depth(\wedge) = 0$ and $depth(i.p) = 1 + depth(p)$; in other words, it is the length of the sequence p. Given a position p with depth n, $p|_k$ is the (prefix) position p at depth $k \leq n$, *i.e.,* $p|_0 = \wedge$, $(i.p)|_k = i.(p|_{k-1})$ if $k > 0$. For instance, for $p = 1.2.1.3$, $p|_3 = 1.2.1$. Given a position p with depth n, $(p)_k$ is the index at the depth $k \leq n$, *i.e.,* $(i.p)_1 = i$ and $(i.p)_{k+1} = (p)_k$.

Definition 10 (Commutative Pair of Subterms). *Given terms* t *and* t' *such that every symbol in* t *and* t' *is either free or commutative, the pair* (u, v) *of terms is called a* commutative pair of subterms of t and t' *if and only if there are positions* $p \in Pos(t)$ *and* $p' \in Pos(t')$ *such that:*

- $t|_p = u$, $t'|_{p'} = v$, $depth(p) = depth(p')$,
- *for each* $0 \leq i < depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$, *and*
- *for each* $0 < j \leq depth(p)$*:*
  - *if* $root(t|_{p|_{j-1}})$ *is free, then* $(p)_j = (p')_j$, *and*
  - *if* $root(t|_{p|_{j-1}})$ *is commutative,* $(p)_j = (p')_j$ *or* $(p)_j = ((p')_j \bmod 2) + 1$.

Example 9. *Let* $t = f_1(f_2(f(a, b)))$ *and* $t' = f_1(f_2(f(b, a))$ *be two terms where* f_1, f_2 *are free and* f *is commutative, i.e.,* $ax(f_1) = ax(f_2) = \emptyset$ *and* $ax(f) = \{C_{f_3}\}$. *The following pairs of terms are commutative subterms of* t *and* t'*:* (t, t'),

$(t|_1, t'|_1)$, $(t|_{1.1.1}, t'|_{1.1.1}) = (a, b)$, $(t|_{1.1.1}, t'|_{1.1.2}) = (a, a)$, $(t|_{1.1.2}, t'|_{1.1.1}) = (b, b)$, *and* $(t|_{1.1.2}, t'|_{1.1.2}) = (b, a)$. *The other possible pairs are not commutative, such as, e.g.,* $(t|_1, t'|_{1.1})$.

Lemma 14. *Given terms* t *and* t' *such that every symbol in* t *and* t' *is either free or commutative, and a fresh variable* x*, then there is a sequence* $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle u \stackrel{y}{\triangleq} v \wedge C \mid S \mid \theta\rangle$ *using the inference rules of Figures 6 and 7 such that there is no variable* z *such that* $u \stackrel{z}{\triangleq} v \in S$ *if and only if* (u, v) *is a commutative pair of subterms of* t *and* t'.

Proof. Straightforward by successive applications of the inference rule $\mathbf{Decompose}_E$ of Figure 6 and the inference rule $\mathbf{Decompose}_C$ of Figure 7. □

Definition 11 (Commutative Conflict Pair). *Given terms* t *and* t' *such that every symbol in* t *and* t' *is either free or commutative, the pair* (u, v) *is called a* commutative conflict pair of t and t' *if and only if* $root(u) \neq root(v)$ *and* (u, v) *is a commutative pair of subterms of* t *and* t'.

The following lemma expresses the appropriate connection between the constraints in a derivation and the commutative conflict pairs of the initial configuration.

Lemma 15. *Given terms* t *and* t' *such that every symbol in* t *and* t' *is either free or commutative, and a fresh variable* x*, then there is a sequence* $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle C \mid u \stackrel{y}{\triangleq} v \wedge S \mid \theta\rangle$ *using the inference rules of Figures 6 and 7 if and only if* (u, v) *is a commutative conflict pair of* t *and* t'.

Proof. ($\Rightarrow$) Since $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle C \mid u \stackrel{y}{\triangleq} v \wedge S \mid \theta\rangle$, then there must be two configurations $\langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$, $\langle C_2 \mid u \stackrel{y}{\triangleq} v \wedge S_2 \mid \theta_2\rangle$ such that

$$\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \to^* \langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle,$$

$$\langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle \to_{\mathbf{Solve}_E} \langle C_2 \mid u \stackrel{y}{\triangleq} v \wedge S_2 \mid \theta_2\rangle,$$

$$\langle C_2 \mid u \stackrel{y}{\triangleq} v \wedge S_2 \mid \theta_2\rangle \to^* \langle \emptyset \mid u \stackrel{y}{\triangleq} v \wedge S \mid \theta\rangle,$$

and, by application of the inference rule $\mathbf{Solve}_E$, $root(u) \neq root(v)$. By using Lemma 14 with the derivation $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \to^* \langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$, (u, v') is a commutative pair of subterms of t and t'. Therefore, (u, v) is a commutative conflict pair.

($\Leftarrow$) By Lemma 14, there is a configuration $\langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$ such that $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \to^* \langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle$, and $root(u) \neq root(v)$. Then, the inference rule $\mathbf{Solve}_E$ is applied, *i.e.,* $\langle u \stackrel{y}{\triangleq} v \wedge C_1 \mid S_1 \mid \theta_1\rangle \to \langle C_1 \mid u \stackrel{y}{\triangleq} v \wedge S_1 \mid \theta_1\rangle$ and $u \stackrel{y}{\triangleq} v$ will be part of S in the final configuration $\langle \emptyset \mid S \mid \theta\rangle$. □

The following lemma establishes the link between the computed substitution and a proper generalizer.

Lemma 16. *Given terms t and t' such that every symbol in t and t' is either free or commutative, and a fresh variable x,*

- *if $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figures 6 and 7, then $x\theta$ is a generalizer of t and t' modulo commutativity;*
- *if u is a generalizer of t and t' modulo commutativity, then there is a derivation $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figures 6 and 7 such that $u \equiv_E x\theta$.*

Proof. By structural induction on the term $x\theta$ (resp. u). If $x\theta = x$ (resp. u is a variable), then $\theta = id$ and the conclusion follows. If f is free and $x\theta = f(u_1, \ldots, u_k)$ (resp. $u = f(u_1, \ldots, u_k)$), then the inference rule $\mathbf{Decompose}_E$ of Figure 6 is applied and we have that $t = f(t_1, \ldots, t_k)$ and $t' = f(t'_1, \ldots, t'_k)$. If $x\theta = f(u_1, \ldots, u_2)$ (resp. $u = f(u_1, \ldots, u_2)$) and f is commutative, then the inference rule $\mathbf{Decompose}_C$ of Figure 7 is applied and we have that either: (i) $t = f(t_1, t_2)$ and $t' = f(t'_1, t'_2)$, or (ii) $t = f(t_1, t_2)$ and $t' = f(t'_2, t'_1)$, or (iii) $t = f(t_2, t_1)$ and $t' = f(t'_1, t'_2)$, or (iv) $t = f(t_2, t_1)$ and $t' = f(t'_2, t'_1)$. For the case where f is free, by using the induction hypothesis, u_i is a generalizer of t_i and t'_i, for each i. For the case where f is commutative, by using the induction hypothesis, u_1 is a generalizer of either t_1 and t'_1, t_1 and t'_2, t_2 and t'_1, or t_2 and t'_2; similarly for u_2. Now, if for each pair of terms in $u_1, \ldots, u_k$ there are no shared variables, then the conclusion follows. Otherwise, for each variable z shared between two different terms u_i and u_j, there is a constraint $w_1 \overset{z}{\triangleq} w_2 \in S$ and, by Lemma 15, there is a commutative conflict pair (w_1, w_2) in t_i and t'_i. Thus, the conclusion follows. □

In the equational case, by correctness we mean that every solution t computed by the equational least general generalization algorithm is an equational least general generalizer for the given problem, whereas completeness means that for every solution t to the generalization problem, a generalizer that is at most as general as t modulo E is computed by the algorithm. From a (finite) complete set of E-generalizers, note that we can always distill a correct set of E-generalizers by just filtering the minimal elements w.r.t. the order $\leq_E$. Correctness and completeness are proved as follows. First, we prove that every term computed by the equational least general generalization algorithm is a generalizer; then we prove that the set of computed generalizers is complete. After filtering away all non-minimal generalizers from the computed set, correctness trivially follows.

Theorem 12 (Correctness). *Given an equational theory (Σ, E), Σ-terms t and t' such that every symbol in t and t' is either free or commutative, and a fresh variable x, if $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle \emptyset \mid S \mid \theta\rangle$ using the inference rules of Figures 6 and 7, then $x\theta$ is a generalizer of t and t' modulo commutativity.*

PROOF. By Lemma 16. □

Theorem 13 (Completeness). *Given an equational theory (Σ, E), Σ-terms t and t' such that every symbol in t and t' is either free or commutative, and a fresh variable x, if u is a least general generalizer of t and t' modulo commutativity, then there is a derivation $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figures 6 and 7, such that $u \equiv_E x\theta$.*

PROOF. By contradiction. Consider a derivation $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle \emptyset \mid S \mid \theta\rangle$ such that $x\theta$ is not a least general generalizer of t and t' up to renaming. Since $x\theta$ is a generalizer of t and t' by Lemma 16, there is a substitution ρ which is not a variable renaming such that $x\theta\rho =_E u$. By Lemma 13, $VRan(\theta) = Var(x\theta)$, hence we can choose ρ with $Dom(\rho) = Var(x\theta)$. Now, since ρ is not a variable renaming, either:

1. there are variables $y, y' \in Var(x\theta)$ and a variable z such that $y\rho = y'\rho = z$, or
2. there is a variable $y \in Var(x\theta)$ and a non-variable term v such that $y\rho = v$.

In case (1), there are two conflict positions p, p' for t and t' such that $u|_p = z = u|_{p'}$ and $x\theta|_p = y$ and $x\theta|_{p'} = y'$. In particular, this means that $t|_p = t|_{p'}$ and $t'|_p = t'|_{p'}$. But this is impossible by Lemmas 15 and 13. In case (2), there is a position p such that $x\theta|_p = y$ and p is neither a conflict position of t and t' nor it is under a conflict position of t and t'. Since this is impossible by Lemmas 15 and 13, the claim is proved. □

We recall again that in general the inference rules of Figures 6 and 7 together are not confluent, and different final configurations $\langle \emptyset \mid S_1 \mid \theta_1\rangle, \ldots, \langle \emptyset \mid S_n \mid \theta_n\rangle$ correspond to different generalizers $x\theta_1, \ldots, x\theta_n$.

5.4. Least general generalization modulo A

In this section we provide a specific inference rule $\boldsymbol{Decompose}_A$ 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 $\boldsymbol{Decompose}_{A-\text{left}}$ and $\boldsymbol{Decompose}_{A-\text{right}}$ rules are given in Figure 8. We will write $\boldsymbol{Decompose}_A$ to denote any of these two rules. We use flattened versions of the terms which use poly-variadic versions of the associative symbols, *i.e.,* being f an associative symbol, with n arguments, and $n \geq 2$, flattened terms are canonical forms w.r.t. the set of rules given by the following rule schema

$$f(x_1, \ldots, f(t_1, \ldots, t_n), \ldots, x_m) \to f(x_1, \ldots, t_1, \ldots, t_n, \ldots, x_m) \quad n, m \geq 2$$

Given an associative symbol f and a term $f(t_1, \ldots, t_n)$ we call *f-alien terms* (or simply *alien terms*) those terms among the $t_1, \ldots, t_n$ that are not rooted by f. In the following, for f an associative poly-varyadic symbol, by convention $f(t)$ denotes the term t itself, since the symbol f needs at least two arguments. The

Decompose$_{A-\text{left}}$

$$\frac{A_f \in ax(f) \wedge C_f \notin ax(f) \wedge n \geq 2m \geq 2 \wedge \wedge k \in \{1,\ldots,n-1\}}{\begin{array}{c}\langle f(t_1,\ldots,t_n) \stackrel{x}{\triangleq} f(t'_1,\ldots,t'_m) \wedge C \mid S \mid \theta\rangle \rightarrow \\ \langle f(t_1,\ldots,t_k) \stackrel{x_1}{\triangleq} t'_1 \wedge f(t_{k+1},\ldots,t_n) \stackrel{x_2}{\triangleq} f(t'_2,\ldots,t'_m) \wedge C \mid S \mid \theta\sigma\rangle\end{array}}$$

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

Decompose$_{A-\text{right}}$

$$\frac{A_f \in ax(f) \wedge C_f \notin ax(f) \wedge n \geq 2 \wedge m \geq 2 \wedge k \in \{1,\ldots,m-1\}}{\begin{array}{c}\langle f(t_1,\ldots,t_n) \stackrel{x}{\triangleq} f(t'_1,\ldots,t'_m) \wedge C \mid S \mid \theta\rangle \rightarrow \\ \langle t_1 \stackrel{x_1}{\triangleq} f(t'_1,\ldots,t'_k) \wedge f(t_2,\ldots,t_n) \stackrel{x_2}{\triangleq} f(t'_{k+1},\ldots,t'_m) \wedge C \mid S \mid \theta\sigma\rangle\end{array}}$$

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

Figure 8: Decomposition rules for an associative (non–commutative) function symbol f

inference rules of Figure 8 replace the syntactic decomposition inference rule for the case of an associative function symbol f, where all *prefixes* of $t_1,\ldots,t_n$ and $t'_1,\ldots,t'_m$ are considered. Note that this rule is (don't know) non-deterministic, hence all possibilities must be explored.

These inference rules for associativity are better than generating all terms in the corresponding equivalence class, as explained in Section 5, since we will eagerly stop the computation whenever we find a constraint $t \stackrel{x}{\triangleq} f(t_1,\ldots,t_n)$ such that $root(t) \neq f$ without considering all the combinations in the equivalence class of $f(t_1,\ldots,t_n)$.

The following example illustrates least general generalization modulo A.

Example 10. *Let $t = f(f(a,c),b)$ and $t' = f(c,b)$ be two terms with f associative, i.e., $ax(f) = \{A_f\}$. Starting from the initial configuration $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle$, 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 obtain that the lgg modulo A of t and t' is $f(x_3,b)$. The computation trace is shown in Figure 9.*

Note that in the example above there is a unique lgg modulo A, although this is not true for some generalization problems as witnessed by the following example.

Example 11. *Let $t = f(f(a,a),f(b,b))$ and $t' = f(f(b,b),b)$ be two terms where f is associative, i.e., $ax(f) = \{A_f\}$. Starting from the initial configuration $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle$, by applying the rules **Solve**$_E$, **Recover**$_E$, and **Decompose**$_A$ above, we end in two terminal configurations $\langle \emptyset \mid S_1 \mid \theta_1\rangle$ and $\langle \emptyset \mid S_2 \mid \theta_2\rangle$, where $\theta_1 = \{x \mapsto f(f(x,x),y)\}$ and $\theta_2 = \{x \mapsto f(f(y,b),b)\}$. Both $x\theta_1$ and $x\theta_2$ are lggs modulo associativity, since they are incomparable modulo associativity.*

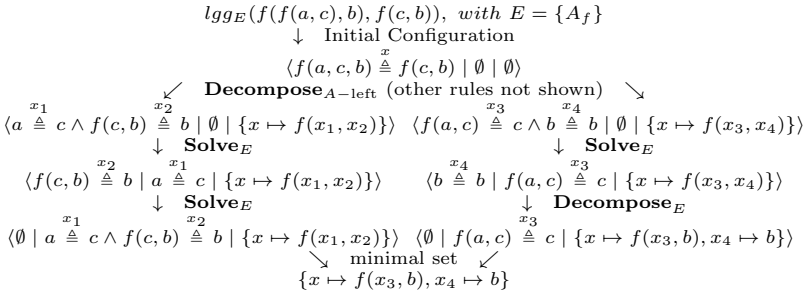

Figure 9: Computation trace for A–generalization of terms $f(f(a,c),b)$ and $f(c,b)$).

Note that the least general generalizer of terms $f(a,c,d,b)$ and $f(a,e,e,b)$ is $f(a,x_1,x_2,b)$ instead of $f(a,x_1,b)$, which may seem the most natural choice. Only when the number of elements is different, a variable takes care of one element of the shortest list and the remaining elements of the longer list, *e.g.,* the least general generalizer of terms $f(a,c,d,b)$ and $f(a,e,e,e,e,b)$ is again $f(a,x_1,x_2,b)$, where x_2 takes care of d and $f(e,e,e)$.

Termination is straightforward.

Theorem 14 (Termination). *Given an equational theory (Σ, E), Σ-terms t and t' such that every symbol in t and t' is either free or associative, and a fresh variable x, every derivation stemming from an initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle$ using the inference rules of Figures 6 and 8, terminates in a final configuration of the form $\langle \emptyset \mid S \mid \theta \rangle$.*

Proof. Similar to the proof of Theorem 1 by simply considering the flattened versions of the terms. □

In order to prove correctness and completeness of the lgg calculus modulo A, we introduce the auxiliary concepts of an *associative pair of subterms* and an *associative conflict pair*, and prove some related, auxiliary results.

First, we prove an auxiliary result stating that only (independently) fresh variables y appear in the index positions of the constraints in C and S components of lgg configurations.

Lemma 17 (Uniqueness of Generalization Variables). *Lemma 1 holds for $t \overset{x}{\triangleq} t'$ when the symbols in t, t' are free or associative, for the inference rules of Figures 6 and 8.*

The lemma below states that the range of the substitutions partially computed at any stage of a generalization derivation coincides with the set of the index variables of the configuration, except for the generalization variable x of the original generalization problem $t \overset{x}{\triangleq} t'$.

Lemma 18 (Range of Substitutions). *Lemma 2 holds for* $t \overset{x}{\triangleq} t'$ *when the symbols in* t, t' *are free or associative, for the inference rules of Figures 6 and 8.*

The following definition establishes an auxiliary property that is useful for defining the notion of an associative conflict pair of terms. Note that the notation $p|_i$ for accessing the symbol at depth i of the position p of a term t is still valid for flattened terms.

Definition 12 (Associative Pair of Positions). *Given flattened terms* t *and* t' *such that every symbol in* t *and* t' *is either free or associative, and given positions* $p \in Pos(t)$ *and* $p' \in Pos(t')$*, the pair* (p, p') *of positions is called an* associative pair of positions of t and t' *if and only if*

- $depth(p) = depth(p')$,
- *for each* $0 \leq i < depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$, *and*
- *for each* $0 < j \leq depth(p)$*:*
  - *if* $root(t|_{p|_{j-1}})$ *is free, then* $(p)_j = (p')_j$, *and*
  - *if* $root(t|_{p|_{j-1}})$ *is associative, then no restriction on* $(p)_j$ *and* $(p')_j$.

Example 12. *Let* $t = f_1(f(a, b, c))$ *and* $t' = f_1(f(d, e))$ *be two flattened terms where* f_1 *is free and* f *is associative, i.e.,* $ax(f_1) = \emptyset$ *and* $ax(f) = \{A_f\}$. *The only possible associative pairs of positions of* t *and* t' *are:* (Λ, Λ), $(1, 1)$, $(1.1, 1.1)$, $(1.1, 1.2)$, $(1.2, 1.1)$, $(1.2, 1.2)$, $(1.3, 1.1)$, *and* $(1.3, 1.2)$. *Any other pair of positions is not associative, such as* $(\Lambda, 1)$ *or* $(1, 1.1)$.

Definition 13 (Associative Pair of Subterms). *Given flattened terms* t *and* t' *such that every symbol in* t *and* t' *is either free or associative, the pair* (u, v) *of terms is called an* associative pair of subterms of t and t' *if and only if either*

1. *(Regular subterms) for each pair of positions* $p \in Pos(t)$ *and* $p' \in Pos(t')$ *such that* $t|_p = u$, $t'|_{p'} = v$, *then* (p, p') *is an associative pair of positions of* t *and* t'*; or*
2. *(Associative subterms) there are positions* $p \in Pos(t)$, $p' \in Pos(t')$ *such that the following conditions are satisfied:*
   - (p, p') *is an associative pair of positions of* t *and* t',
   - $u = f(u_1, \ldots, u_{n_u})$, $n_u \geq 1$, $v = f(v_1, \ldots, v_{n_v})$, $n_v \geq 1$, f *is associative,*
   - $t|_p = f(t_1, \ldots, t_{k_1}, u_1, \ldots, u_{n_u}, t_{k_2}, \ldots, t_{n_p})$, $n_p \geq 2$, $t'|_{p'} = f(t'_1, \ldots, t'_{k'_1}, v_1, \ldots, v_{n_v}, t'_{k'_2}, \ldots, t'_{n_{p'}})$, $n_p \geq 2$, *and*
   - $k_1 = 0$ *(no arguments before* u_1*) if and only if* $k'_1 = 0$ *(no arguments before* v_1*), and,*
   - $k_2 > n_p$ *(no arguments after* u_{n_u}*) if and only if* $k'_2 > n_{p'}$ *(no arguments after* v_{n_v}*).*

Example 13. *Consider again Example 12. Let* $t = f_1(f(a,b,c))$ *and* $t' = f_1(f(d,e))$ *be two flattened terms where* f_1 *is free and* f *is associative, i.e.,* $ax(f_1) = \emptyset$ *and* $ax(f) = \{A_f\}$. *Some associative pairs of subterms of* t *and* t' *are the following, corresponding to associative pairs of positions of* t *and* t'*:* $(t|_\Lambda, t'|_\Lambda)$, $(t|_1, t'|_1)$, $(t|_{1.1}, t'|_{1.1}) = (a,d)$, $(t|_{1.1}, t'|_{1.2}) = (a,e)$, $(t|_{1.2}, t'|_{1.1}) = (b,d)$, $(t|_{1.2}, t'|_{1.2}) = (b,e)$, $(t|_{1.3}, t'|_{1.1}) = (c,d)$, $(t|_{1.3}, t'|_{1.2}) = (c,e)$. *But we also consider the following pairs of subterms as associative pairs:* $(f(a,b),d)$ *and* $(f(b,c),e)$*; in both pairs of subterms, the associative pair of positions is* $(1,1)$. *Note that the pairs of terms* $(f(a,b),e)$ *and* $(f(b,c),d)$ *are not valid associative pairs.*

The following lemma establishes an auxiliary property that is useful for defining the notion of an associative conflict pair of terms.

Lemma 19. *Given flattened terms* t *and* t' *such that every symbol in* t *and* t' *is either free or associative, and a fresh variable* x*, then there is a sequence* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle u \overset{y}{\triangleq} v \wedge C \mid S \mid \theta\rangle$ *using the inference rules of Figures 6 and 8 such that there is no variable* z *such that* $u \overset{z}{\triangleq} v \in S$ *if and only if* (u,v) *is an associative pair of subterms of* t *and* t'.

Proof. Straightforward by successive applications of the inference rule $\mathbf{Decompose}_E$ of Figure 6 and the inference rules $\mathbf{Decompose}_{A-\text{left}}$ and $\mathbf{Decompose}_{A-\text{right}}$ of Figure 8. □

Definition 14 (Associative Conflict Pair). *Given flattened terms* t *and* t' *such that every symbol in* t *and* t' *is either free or associative, the pair* (u,v) *is called an* associative conflict pair of t and t' *if and only if* $root(u) \neq root(v)$ *and* (u,v) *is an associative pair of subterms of* t *and* t'.

The following lemma expresses the appropriate connection between the constraints in a derivation and the associative conflict pairs of the initial configuration.

Lemma 20. *Given flattened terms* t *and* t' *such that every symbol in* t *and* t' *is either free or associative, and a fresh variable* x*, then there is a sequence* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle C \mid u \overset{y}{\triangleq} v \wedge S \mid \theta\rangle$ *using the inference rules of Figures 6 and 8 if and only if* (u,v) *is an associative conflict pair of* t *and* t'.

Proof. Similar to the proof of Lemma 15 but using associative conflict pairs and Lemma 19 instead of commutative conflict pairs and Lemma 14. □

Finally, the following lemma establishes the link between the computed substitution and a proper generalizer.

Lemma 21. *Given flattened terms* t *and* t' *such that every symbol in* t *and* t' *is either free or associative, and a fresh variable* x*,*

- *if* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ *using the inference rules of Figures 6 and 8, then* $x\theta$ *is a generalizer of* t *and* t' *modulo associativity;*
- *if* u *is a generalizer of* t *and* t' *modulo associativity, then there is a derivation* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ *using the inference rules of Figures 6 and 8, such that* $u \equiv_E x\theta$.

PROOF. By structural induction on the term $x\theta$ (resp. u). If $x\theta = x$ (resp. u is a variable), then $\theta = id$ and the conclusion follows. If f is free and $x\theta = f(u_1, \ldots, u_k)$ (resp. $u = f(u_1, \ldots, u_k)$), then the inference rule **Decompose**$_E$ of Figure 6 is applied and we have that $t = f(t_1, \ldots, t_k)$ and $t' = f(t'_1, \ldots, t'_k)$. If f is associative and $x\theta = f(u_1, \ldots, u_k)$ (resp. $u = f(u_1, \ldots, u_k)$) is a flattened term, then the inference rules **Decompose**$_A$ of Figure 8 are applied and we have that $t = f(t_1, \ldots, t_n)$ and $t' = f(t'_1, \ldots, t'_m)$. For the case where f is free, by using the induction hypothesis, u_i is a generalizer of t_i and t'_i, for each i. For the case where f is associative, by using the induction hypothesis, u_1 is a generalizer of a prefix of t and a prefix of t', *i.e.,* u_1 is a generalizer of $f(t_1, \ldots, t_i)$, $1 \leq i \leq n-1$ and $f(t'_1, \ldots, t'_j)$, $1 \leq j \leq m-1$. Similarly, u_k is a generalizer of a postfix of t and t', and the remaining terms u_j , $1 < j < k$, are generalizers of subsequences of t and t', respecting the order between u_j and u_{j+1} for each j. It is easy to see that successive applications of the inference rules **Decompose**$_A$ of Figure 8 consider all possible combinations of prefixes, subsequences, or postfixes determined before. Now, if for each pair of terms in $u_1, \ldots, u_k$ there are no shared variables, then the conclusion follows. Otherwise, for each variable z shared between two different terms u_i and u_j, there is a constraint $w_1 \overset{z}{\triangleq} w_2 \in S$ and, by Lemma 20, there is an associative conflict pair (w_1, w_2) in t_i and t'_i. Thus, the conclusion follows. □

Finally, correctness and completeness are proved as follows.

Theorem 15 (Correctness). *Given an equational theory* (Σ, E)*, flattened* Σ*-terms* t *and* t' *such that every symbol in* t *and* t' *is either free or associative, and a fresh variable* x*, if* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ *using the inference rules of Figures 6 and 8, then* $x\theta$ *is a generalizer of* t *and* t' *modulo associativity.*

PROOF. By Lemma 21. □

Theorem 16 (Completeness). *Given an equational theory* (Σ, E)*, flattened* Σ*-terms* t *and* t' *such that every symbol in* t *and* t' *is either free or associative, and a fresh variable* x*, if* u *is a least general generalizer of* t *and* t' *modulo associativity, then there is a derivation* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ *using the inference rules of Figures 6 and 8 such that* $u \equiv_E x\theta$.

PROOF. Similar to the proof of Theorem 13 but using Lemmas 18, 20, 21 instead of Lemmas 13, 15, 16. □

Decompose$_{AC-\text{left}}$

$$\frac{\{A_f, C_f\} \subseteq ax(f) \wedge n \geq 2 \wedge m \geq 2 \wedge \{i_1, \ldots, i_n\} = \uplus\{1, \ldots, n\} \wedge k_n \in \{1, \ldots, n-1\} \wedge k_m \in \{1, \ldots, m\}}{\begin{array}{c}\langle f(t_1, \ldots, t_n) \stackrel{x}{\triangleq} f(t'_1, \ldots, t'_m) \wedge C \mid S \mid \theta\rangle \rightarrow \\ \langle f(t_{i_1}, \ldots, t_{i_{k_n}}) \stackrel{x_1}{\triangleq} t'_{k_m} \wedge \\ f(t_{i_{(k_n+1)}}, \ldots, t_{i_n}) \stackrel{x_2}{\triangleq} f(t'_1, \ldots, t'_{k_m-1}, t'_{k_m+1}, \ldots, t'_m) \wedge C \mid S \mid \theta\sigma\rangle\end{array}}$$

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

Decompose$_{AC-\text{right}}$

$$\frac{\{A_f, C_f\} \subseteq ax(f) \wedge n \geq 2 \wedge m \geq 2 \wedge \{i_1, \ldots, i_m\} = \uplus\{1, \ldots, m\} \wedge k_m \in \{1, \ldots, m-1\} \wedge k_n \in \{1, \ldots, n\}}{\begin{array}{c}\langle f(t_1, \ldots, t_n) \stackrel{x}{\triangleq} f(t'_1, \ldots, t'_m) \wedge C \mid S \mid \theta\rangle \rightarrow \\ \langle t_{k_n} \stackrel{x_1}{\triangleq} f(t'_{i_1}, \ldots, t'_{i_{k_m}}) \wedge \\ f(t_1, \ldots, t_{k_n-1}, t_{k_n+1}, \ldots, t_n) \stackrel{x_2}{\triangleq} f(t'_{i_{(k_m+1)}}, \ldots, t'_{i_m}) \wedge C \mid S \mid \theta\sigma\rangle\end{array}}$$

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

Figure 10: Decomposition rules for an associative–commutative function symbol f

We recall again that in general the inference rules of Figures 6 and 8 together are not confluent, and different final configurations $\langle \emptyset \mid S_1 \mid \theta_1\rangle, \ldots, \langle \emptyset \mid S_n \mid \theta_n\rangle$ correspond to different generalizers $x\theta_1, \ldots, x\theta_n$.

5.5. 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 5.4. Actually, by considering AC function symbols as varyadic functions with no ordering among the arguments, an AC term can be represented by a canonical representative (Eker, 2003; Hullot, 1980) such that $=_{AC}$ is decidable.

The new decomposition rules for the AC case are similar to the two decompose inference rules for associative function symbols, except that all permutations of $f(t_1, \ldots, t_n)$ and $f(s_1, \ldots, s_m)$ are considered.

To simplify, we write $\{i_1, \ldots, i_n\} = \uplus\{1, \ldots, n\}$ to denote that the sequence $\{i_1, \ldots, i_n\}$ is a permutation of the sequence.

Example 14. *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., $ax(f) = \{A_f, C_f\}$. Starting from the initial configuration $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle$, by applying the rules* ***Solve***$_E$, ***Recover***$_E$, *and* ***Decompose***$_{AC-\text{left}}$, *above, we end in two terminal configurations whose*

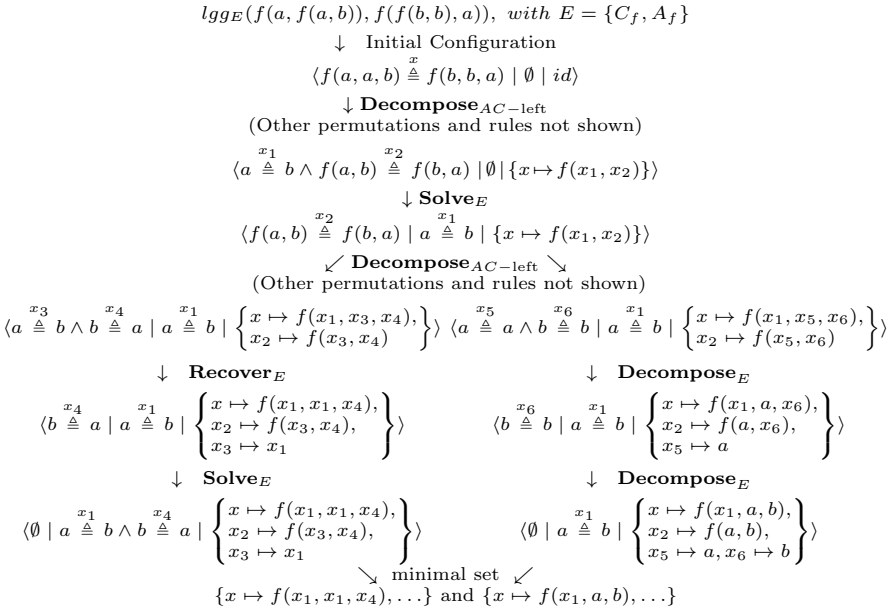

Figure 11: Computation trace for AC–generalizations of terms $f(a, f(a,b))$ and $f(f(b,b),a)$.

respective substitution components are $\theta_1 = \{x \mapsto f(x_1,x_1,x_4), x_2 \mapsto f(x_3,x_4), x_3 \mapsto x_1\}$ *and* $\theta_2 = \{x \mapsto f(x_1,a,b), x_2 \mapsto f(a,b), x_5 \mapsto a, x_6 \mapsto b\}$*, thus we compute that the lggs modulo AC of* t *and* s *are* $f(x_1,x_1,x_4)$ *and* $f(x_1,a,b)$*. The corresponding computation trace is shown in Figure 11.*

Termination is straightforward.

Theorem 17 (Termination). *Given an equational theory* (Σ, E)*,* Σ*-terms* t *and* t' *such that every symbol in* t *and* t' *is either free or associative-commutative, and a fresh variable* x*, every derivation stemming from an initial configuration* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle$ *using the inference rules of Figures 6 and 10 terminates in a final configuration of the form* $\langle \emptyset \mid S \mid \theta \rangle$*.*

Proof. Similar to the proof of Theorem 1 by simply considering the flattened versions of the terms. □

In order to prove correctness and completeness of the lgg calculus modulo associativity and commutativity, we introduce the auxiliary concepts of an *associative-commutative pair of subterms* and an *associative-commutative conflict pair*, and prove some related, auxiliary results.

First, we prove an auxiliary result stating that only (independently) fresh variables y appear in the index positions of the constraints in C and S components of lgg configurations.

Lemma 22 (Uniqueness of Generalization Variables). *Lemma 1 holds for $t \overset{x}{\triangleq} t'$ when the symbols in t, t' are free or associative-commutative, for the inference rules of Figures 6 and 10.*

The lemma below states that the range of the substitutions partially computed at any stage of a generalization derivation coincides with the set of the index variables of the configuration, except for the generalization variable x of the original generalization problem $t \overset{x}{\triangleq} t'$.

Lemma 23 (Range of Substitutions). *Lemma 2 holds for $t \overset{x}{\triangleq} t'$ when the symbols in t, t' are free or associative-commutative, for the inference rules of Figures 6 and 10.*

Here, we reuse the notion of associative pair of positions and do not have to provide a new notion of associative-commutative pair of positions.

Definition 15 (Associative-commutative Pair of Positions). *Given flattened terms t and t' such that every symbol in t and t' is either free or associative-commutative, and given positions $p \in Pos(t)$ and $p' \in Pos(t')$, the pair (p, p') of positions is called an* associative-commutatve pair of positions of t and t' *if it satisfies the conditions for being an associative pair of positions of t and t'.*

But we do provide a new notion of associative-commutative pair of subterms, different from that of associative pair of subterms.

Definition 16 (Associative-commutative Pair of Subterms). *Given flattened terms t and t' such that every symbol in t and t' is either free or associative-commutative, the pair (u, v) of terms is called an* associative-commutative pair of subterms of t and t' *if and only if either*

1. *(Regular subterms) for each pair of positions $p \in Pos(t)$ and $p' \in Pos(t')$ such that $t|_p = u$, $t'|_{p'} = v$, then (p, p') is an associative-commutative pair of positions of t and t'; or*
2. *(Associative-commutative subterms) there are positions $p \in Pos(t)$, $p' \in Pos(t')$ such that the following conditions are satisfied:*
   - *(p, p') is an associative-commutative pair of positions of t and t', and*
   - *$u = f(u_1, \ldots, u_{n_u})$, $n_u \geq 1$, $v = f(v_1, \ldots, v_{n_v})$, $n_v \geq 1$, f is associative,*
   - *$t|_p = f(t_1, \ldots, t_{n_p})$, $n_p \geq 2$, $t'|_{p'} = f(t'_1, \ldots, t'_{n_{p'}})$, $n_{p'} \geq 2$,*
   - *for all $1 \leq i \leq n_u$, there exists $1 \leq j \leq n_p$ s.t. $u_i =_E t_j$, and*
   - *for all $1 \leq i \leq n_v$, there exists $1 \leq j \leq n_{p'}$ s.t. $v_i =_E t'_j$.*

Example 15. *Let $t = f_1(f(a, b, c))$ and $t' = f_1(f(d, e))$ be two flattened terms where f_1 is free and f is associative-commutative, i.e., $ax(f_1) = \emptyset$ and $ax(f) = \{A_f, C_f\}$. Some associative-commutative pairs of subterms of t and t' are the*

following, corresponding to associative-commutative pairs of positions of t and t': $(t|_\Lambda, t'|_\Lambda)$, $(t|_1, t'|_1)$, $(t|_{1.1}, t'|_{1.1}) = (a, d)$, $(t|_{1.1}, t'|_{1.2}) = (a, e)$, $(t|_{1.2}, t'|_{1.1}) = (b, d)$, $(t|_{1.2}, t'|_{1.2}) = (b, e)$, $(t|_{1.3}, t'|_{1.1}) = (c, d)$, $(t|_{1.3}, t'|_{1.2}) = (c, e)$. But we also consider the following pairs of subterms as associative-commutative pairs: $(f(a, b), d)$, $(f(a, c), d)$, $(f(b, c), d)$, $(f(a, b), e)$, $(f(a, c), e)$, $(f(b, c), e)$; in all these pairs of subterms, the associative-commutatvive pair of positions is $(1, 1)$.

The following lemma establishes an auxiliary property that is useful for defining the notion of an associative-commutative conflict pair of terms.

Lemma 24. *Given flattened terms t and t' such that every symbol in t and t' is either free or associative-commutative, and a fresh variable x, then there is a sequence $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle u \overset{y}{\triangleq} v \wedge C \mid S \mid \theta\rangle$ using the inference rules of Figures 6 and 10 such that there is no variable z such that $u \overset{z}{\triangleq} v \in S$ if and only if (u, v) is an associative-commutative pair of subterms of t and t'.*

Proof. Straightforward by successive applications of the inference rule $\mathbf{Decompose}_E$ of Figure 6 and the inference rules $\mathbf{Decompose}_{AC-\text{left}}$ and $\mathbf{Decompose}_{AC-\text{right}}$ of Figure 10. □

Definition 17 (Associative-commutative Conflict Pair). *Given flattened terms t and t' such that every symbol in t and t' is either free or associative-commutative, the pair (u, v) is called an* associative-commutative conflict pair of t and t' *if and only if $root(u) \neq root(v)$ and (u, v) is an associative-commutative pair of subterms of t and t'.*

The following lemma expresses the appropriate connection between the constraints in a derivation and the associative-commutative conflict pairs of the initial configuration.

Lemma 25. *Given flattened terms t and t' such that every symbol in t and t' is either free or associative-commutative, and a fresh variable x, then there is a sequence $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle C \mid u \overset{y}{\triangleq} v \wedge S \mid \theta\rangle$ using the inference rules of Figures 6 and 10 if and only if (u, v) is an associative-commutative conflict pair of t and t'.*

Proof. Similar to the proof of Lemma 20 but using associative-commutative conflict pairs and Lemma 24 instead of associative conflict pairs and Lemma 19. □

Finally, the following lemma establishes the link between the computed substitution and a proper generalizer.

Lemma 26. *Given flattened terms t and t' such that every symbol in t and t' is either free or associative-commutative, and a fresh variable x,*

- *if* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ *using the inference rules of Figures 6 and 10, then* $x\theta$ *is a generalizer of* t *and* t' *modulo associativity and commutativity;*
- *if* u *is a generalizer of* t *and* t' *modulo associativity and commutativity, then there is a derivation* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ *using the inference rules of Figures 6 and 10 such that* $u \equiv_E x\theta$.

PROOF. By structural induction on the term $x\theta$ (resp. u). If $x\theta = x$ (resp. u is a variable), then $\theta = id$ and the conclusion follows. If f is free and $x\theta = f(u_1, \ldots, u_k)$ (resp. $u = f(u_1, \ldots, u_k)$), then the inference rule $\mathbf{Decompose}_E$ of Figure 6 is applied and we have that $t = f(t_1, \ldots, t_k)$ and $t' = f(t'_1, \ldots, t'_k)$. If f is associative and commutative and $x\theta = f(u_1, \ldots, u_k)$ (resp. $u = f(u_1, \ldots, u_k)$) is a flattened term, then the inference rules $\mathbf{Decompose}_{AC}$ of Figure 10 are applied and we have that $t = f(t_1, \ldots, t_n)$ and $t' = f(t'_1, \ldots, t'_m)$. For the case where f is free, by using the induction hypothesis, u_i is a generalizer of t_i and t'_i, for each i. For the case where f is associative and commutative, by using the induction hypothesis, each u_i is a generalizer of a subsequence of one permutation of prefix of t and a prefix of t', *i.e.,* u_1 is a generalizer of t and t'. It is easy to see that successive applications of the inference rules $\mathbf{Decompose}_{AC}$ of Figure 10 consider all possible permutations and subsequences determined before. Now, if for each pair of terms in $u_1, \ldots, u_k$ there are no shared variables, then the conclusion follows. Otherwise, for each variable z shared between two different terms u_i and u_j, there is a constraint $w_1 \overset{z}{\triangleq} w_2 \in S$ and, by Lemma 25, there is an associative-commutative conflict pair (w_1, w_2) in t_i and t'_i. Thus, the conclusion follows. □

Finally, correctness and completeness are proved as follows.

Theorem 18 (Correctness). *Given an equational theory* (Σ, E)*, flattened* Σ*-terms* t *and* t' *such that every symbol in* t *and* t' *is either free or associative-commutative, and a fresh variable* x*, if* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ *using the inference rules of Figures 6 and 10, then* $x\theta$ *is a generalizer of* t *and* t' *modulo associativity and commutativity.*

PROOF. By Lemma 26. □

Theorem 19 (Completeness). *Given an equational theory* (Σ, E)*, flattened* Σ*-terms* t *and* t' *such that every symbol in* t *and* t' *is either free or associative-commutative, and a fresh variable* x*, if* u *is a least general generalizer of* t *and* t' *modulo associativity and commutativity, then there is a derivation* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ *using the inference rules of Figures 6 and 10 such that* $u \equiv_E x\theta$.

PROOF. Similar to the proof of Theorem 16 but using Lemmas 23, 25, 26 instead of Lemmas 18, 20, 21. □

$$\textbf{Expand}_U \quad \frac{root(t) \equiv f \wedge U_f \in ax(f) \wedge t' \neq_E e \wedge root(t') \neq f \wedge \quad t'' \in \{f(e,t'), f(t',e)\}}{\langle t \overset{x}{\triangleq} t' \wedge C \mid S \mid \theta\rangle \rightarrow \langle t \overset{x}{\triangleq} t'' \wedge C \mid S \mid \theta\rangle}$$

Figure 12: Inference rule for expanding function symbol f with identity element e

We recall again that in general the inference rules of Figures 6 and 10 together are not confluent, and different final configurations $\langle \emptyset \mid S_1 \mid \theta_1\rangle, \ldots, \langle \emptyset \mid S_n \mid \theta_n\rangle$ correspond to different generalizers $x\theta_1, \ldots, x\theta_n$.

5.6. Least general generalization modulo U

Finally, let us introduce the inference rule of Figure 12 for handling function symbols f which have an identity element e, *i.e.,* an element such that $f(x,e) \doteq x$ and $f(e,x) \doteq x$. This rule considers the identity axioms in a rather lazy or on-demand manner to avoid infinite generation of all the elements in the equivalence class. The rule corresponds to the case when the root symbol f of the term t in the left–hand side of the constraint $t \overset{x}{\triangleq} s$ has e as an identity element. A companion rule for handling the case when the root symbol f of the term t' in the right–hand side has e as an identity element is omitted, since it is entirely similar.

Example 16. *Let $t = f(a,b,c,d)$ and $s = f(a,c)$ be two terms where $ax(f) = \{A_f, C_f, U_f\}$. Starting from the initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle$, by applying the rules **Solve**$_E$, **Recover**$_E$, **Decompose**$_{AC-\text{left}}$, 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 13.*

Note that in the example above there is a unique lgg modulo U, although this is not true for some generalization problems as witnessed by the following example.

Example 17. *Let $t = f(f(a,a), f(b,a))$ and $t' = f(f(b,b), a)$ be two terms such that $\{A_f, U_f\} \subseteq ax(f)$. Starting from the initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle$, by applying the rules **Solve**$_E$, **Recover**$_E$, **Decompose**$_{A-\text{left}}$, and **Expand**$_U$ above, we end in two terminal configurations $\langle \emptyset \mid S_1 \mid \theta_1\rangle$ and $\langle \emptyset \mid S_2 \mid \theta_2\rangle$, where $\theta_1 = \{x \mapsto f(f(x,x), f(y,a))\}$ and $\theta_2 = \{x \mapsto f(y, f(b,a))\}$. Both $x\theta_1$ and $x\theta_2$ are lggs modulo AU, since they are incomparable modulo AU.*

Termination is slightly more difficult when there are symbols with identities.

Theorem 20 (Termination). *Given an equational theory (Σ, E), Σ-terms t and t' such that every symbol in t and t' is either free or has an identity,*

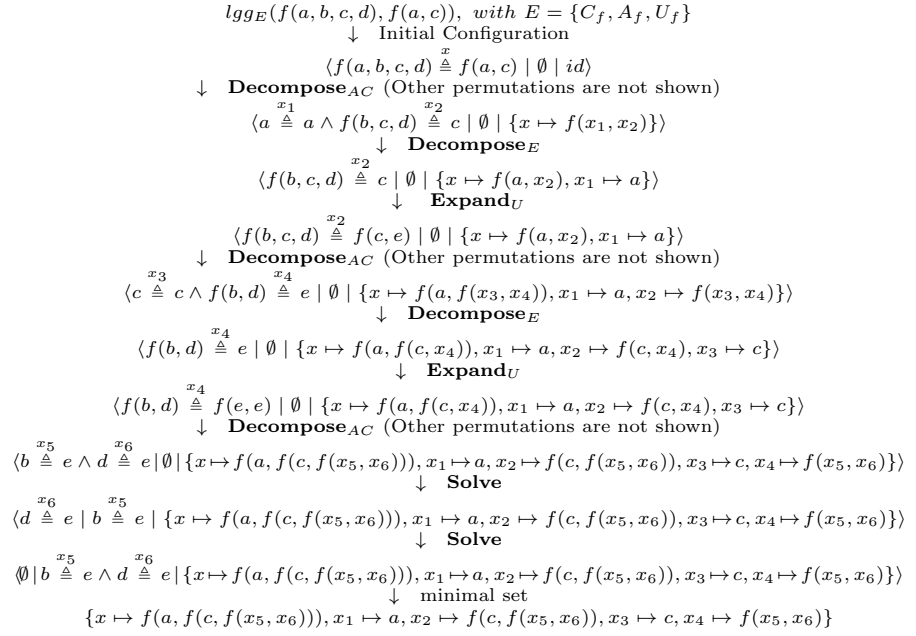

Figure 13: Computation trace for ACU–generalization of terms $f(a,b,c,d)$ and $f(a,c)$.

and a fresh variable x, every derivation stemming from an initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle$ using the inference rules of Figures 6 and 12 terminates in a final configuration of the form $\langle \emptyset \mid S \mid \theta \rangle$.

Proof. Let $|u|$ be the number of symbol occurrences in the syntactic object u. Let k be the minimum of $|t|$ and $|t'|$. k is an upper bound to the number of times that the inference rule $\mathbf{Decompose}_E$ of Figure 6 can be applied. Let $\overline{k}$ be the maximum of $|t|$ and $|t'|$. Since the inference rule $\mathbf{Expand}_U$ adds a symbol f with an identity to one side of a constraint only when the other side already has such a symbol, $\overline{k} - k$ is an upper bound to the number of times that the inference rule $\mathbf{Expand}_U$ followed by a decomposing rule of Figure 6 (or Figures 7, 8, and 10) can be applied. Finally, the application of rules $\mathbf{Solve}_E$ and $\mathbf{Recover}_E$ strictly decreases the size $|C|$ of the C component of the lgg configurations at each step, hence the derivation terminates. □

In order to prove correctness and completeness of the lgg calculus modulo identify, we introduce the auxiliary concepts of an *identify pair of subterms* and an *identity conflict pair*, and prove some related, auxiliary results.

First, we prove an auxiliary result stating that only (independently) fresh variables y appear in the index positions of the constraints in C and S components of lgg configurations.

Lemma 27 (Uniqueness of Generalization Variables). *Lemma 1 holds for $t \overset{x}{\triangleq} t'$ when the symbols in t, t' are free or have some identity symbols, for the inference rules of Figures 6 and 12.*

The lemma below states that the range of the substitutions partially computed at any stage of a generalization derivation coincides with the set of the index variables of the configuration, except for the generalization variable x of the original generalization problem $t \overset{x}{\triangleq} t'$.

Lemma 28 (Range of Substitutions). *Lemma 2 holds for $t \overset{x}{\triangleq} t'$ when the symbols in t, t' are free or have some identity symbols, for the inference rules of Figures 6 and 12.*

The following definition establishes an auxiliary property that is useful for definition of the notion of an identity conflict pair of terms.

Definition 18 (Identity Pair of Positions). *Given terms t and t' such that every symbol in t and t' is either free or has an identity, and given positions $p \in Pos(t)$ and $p' \in Pos(t')$, the pair (p, p') of positions is called an* identity pair of positions of t and t' *if and only if either*

1. *(Base case) $p = \Lambda$ and $p' = \Lambda$;*
2. *(Free symbol) $p = q.i$, $p' = q'.i$, $root(t'|_{q'}) = root(t|_q)$ is a free symbol, and (q, q') is an identity pair of positions of t and t';*
3. *(Identity on the left) $depth(p) > depth(p')$, $p = q.i$, $root(t|_q)$ has an identity symbol e, and (q, p') is an identity pair of positions of t and t'; or*
4. *(Identity on the right) $depth(p') > depth(p)$, $p' = q'.i$, $root(t'|_{q'})$ has an identity symbol e, and (p, q') is an identity pair of positions of t and t'.*

Example 18. *Let $t = f_1(f(a, b))$ and $t' = f_1(c)$ be two flattened terms where f_1 is free and f has an identity, i.e., $ax(f_1) = \emptyset$ and $ax(f) = \{U_f\}$. The only possible identity pairs of positions of t and t' are: (Λ, Λ), $(1, 1)$, $(1.1, 1)$, and $(1.2, 1)$. Any other pair of positions is not identity, such as $(\Lambda, 1)$.*

Definition 19 (Identity Pair of Subterms). *Given terms t and t' such that every symbol in t and t' is either free or has an identity, the pair (u, v) of terms is called an* identity pair of subterms of t and t' *if and only if for each pair of positions $p \in Pos(t)$ and $p' \in Pos(t')$ such that $t|_p = u$, $t'|_{p'} = v$, then (p, p') is an identity pair of positions of t and t'.*

Example 19. *Let $t = f_1(f(a, b))$ and $t' = f_1(c)$ be two flattened terms where f_1 is free and f has an identity symbol e, i.e., $ax(f_1) = \emptyset$ and $ax(f) = \{U_f\}$. The only possible identity pairs of subterms of t and t' are the ones corresponding to identity pairs of positions: $(t|_\Lambda, t'|_\Lambda)$, $(t|_1, t'|_1)$, $(t|_{1.1}, t'|_1) = (a, c)$, and $(t|_{1.2}, t'|_1) = (b, c)$.*

The following lemma establishes an auxiliary property that is useful for defining the notion of an identity conflict pair of terms.

Lemma 29. *Given terms t and t' such that every symbol in t and t' is either free or has an identity, and a fresh variable x, then there is a sequence $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle u \overset{y}{\triangleq} v \wedge C \mid S \mid \theta\rangle$ using the inference rules of Figures 6 and 12 such that there is no variable z such that $u \overset{z}{\triangleq} v \in S$ if and only if (u, v) is an identity pair of subterms of t and t'.*

PROOF. Straightforward by successive application of the inference rule $\mathbf{Decompose}_E$ of Figure 6 and the inference rule $\mathbf{Decompose}_U$ of Figure 12.□

Definition 20 (Identity Conflict Pair). *Given terms t and t' such that every symbol in t and t' is either free or has an identity, the pair (u, v) is called an* identity conflict pair of t and t' *if and only if $root(u) \neq root(v)$ and (u, v) is an identity pair of subterms of t and t'.*

The following lemma expresses the appropriate connection between the constraints in a derivation and the identity conflict pairs of the initial configuration.

Lemma 30. *Given terms t and t' such that every symbol in t and t' is either free or has an identity, and a fresh variable x, then there is a sequence $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid u \overset{y}{\triangleq} v \wedge S \mid \theta\rangle$ using the inference rules of Figures 6 and 12 if and only if (u, v) is an identity conflict pair of t and t'.*

PROOF. Similar to the proof of Lemma 25 but using identity conflict pairs and Lemma 29 instead of associative-commutative conflict pairs and Lemma 24. □

Finally, the following lemma establishes the link between the computed substitution and a proper generalizer.

Lemma 31. *Given terms t and t' such that every symbol in t and t' is either free or has an identity, and a fresh variable x,*

- *if $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figures 6 and 12, then $x\theta$ is a generalizer of t and t' modulo identity;*
- *if u is a generalizer of t and t' modulo identity, then there is a derivation $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figures 6 and 12, such that $u =_E x\theta$.*

PROOF. By structural induction on the term $x\theta$ (resp. u). If $x\theta = x$ (resp. u is a variable), then $\theta = id$ and the conclusion follows. If $x\theta = f(u_1, \ldots, u_k)$ (resp. $u = f(u_1, \ldots, u_k)$) and f is free, then the inference rule $\mathbf{Decompose}_E$ of Figure 6 is applied and we have that $t = f(t_1, \ldots, t_k)$ and $t' = f(t'_1, \ldots, t'_k)$. If f has an identity symbol e and $x\theta = f(u_1, u_2)$ (resp. $u = f(u_1, u_2)$), then the

inference rule **Decompose**$_U$ of Figure 12 is applied and we have that either: (i) $t = f(t_1, t_2)$ and $t' = f(t'_1, t'_2)$, (ii) $t = f(t_1, t_2)$ and $root(t') \neq f$, or (iii) $root(t) \neq f$ and $t' = f(t'_1, t'_2)$. For the case where f is free, by using the induction hypothesis, u_i is a generalizer of t_i and t'_i, for each i. For the case where f has an identity symbol e, by using the induction hypothesis, u_1 is a generalizer of either t_1 and t'_1, t_1 and t' (by applying$f(x, e) \doteq x$ to t'), or t and t'_1 (by applying$f(x, e) \doteq x$ to t). Similarly, u_2 is a generalizer of either t_2 and t'_2, t_2 and t' (by applying$f(e, x) \doteq x$ to t'), or t and t'_2 (by applying$f(e, x) \doteq x$ to t). Now, if for each pair of terms in $u_1, \ldots, u_k$ there are no shared variables, then the conclusion follows. Otherwise, for each variable z shared between two different terms u_i and u_j, there is a constraint $w_1 \overset{z}{\triangleq} w_2 \in S$ and, by Lemma 30, there is an identity conflict pair (w_1, w_2) in t_i and t'_i. Thus, the conclusion follows. □

Finally, correctness and completeness are proved as follows.

Theorem 21 (Correctness). *Given an equational theory (Σ, E), Σ-terms t and t' such that every symbol in t and t' is either free or has an identity, and a fresh variable x, if $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ using the inference rules of Figures 6 and 12, then $x\theta$ is a generalizer of t and t' modulo identity.*

Proof. By Lemma 31. □

Theorem 22 (Completeness). *Given an equational theory (Σ, E), Σ-terms t and t' such that every symbol in t and t' is either free or has an identity, and a fresh variable x, if u is a least general generalizer of t and t' modulo identity, then there is a derivation $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figures 6 and 12 such that $u \equiv_E x\theta$.*

Proof. Similar to the proof of Theorem 19 but using Lemmas 28, 30, 31 instead of Lemmas 23, 25, 26. □

We recall again that in general the inference rules of Figures 6 and 12 together are not confluent, and different final configurations $\langle \emptyset \mid S_1 \mid \theta_1\rangle, \ldots, \langle \emptyset \mid S_n \mid \theta_n\rangle$ correspond to different generalizers $x\theta_1, \ldots, x\theta_n$. But note that if a symbol f has an identity element e and is commutative or associative-commutative, then it is not necessary to consider both forms $f(t', e)$ and $f(e, t')$ in Figure 12 above.

5.7. A modular ACU-generalization method

For the general case when different function symbols satisfying different associativity and/or commutativity and/or identity axioms are considered, we can use the inference rules above all together (inference rules of Figures 6, 7, 8, 10, and 12) 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 algorithm is straightforward.

Theorem 23 (Termination). *For an equational theory (Σ, E) with $E \in \mathbb{E}$, Σ-terms t and t', and a fresh variable x, every derivation stemming from an initial configuration $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle$ using the inference rules of Figures 6, 7, 8, 10, and 12, terminates in a final configuration of the form $\langle \emptyset \mid S \mid \theta \rangle$.*

The correctness and completeness of our algorithm is ensured by the following two results.

Theorem 24 (Correctness). *For an equational theory (Σ, E) with $E \in \mathbb{E}$, Σ-terms t and t', and a fresh variable x, if $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle \emptyset \mid S \mid \theta \rangle$ using the inference rules of Figures 6, 7, 8, 10, and 12, then $x\theta$ is a generalizer of t and t' modulo E.*

Theorem 25 (Completeness). *For an equational theory (Σ, E) with $E \in \mathbb{E}$, Σ-terms t and t', and a fresh variable x, if u is a least general generalizer of t and t' modulo E, then there is a derivation $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle C \mid S \mid \theta \rangle$ using the inference rules of Figures 6, 7, 8, 10, and 12, such that $u \equiv_E x\theta$.*

6. Order-Sorted Least General Generalizations modulo E

In this section, we generalize the unsorted modular equational generalization algorithm presented in Section 5 to the order-sorted setting.

First of all, pre-regularity is not enough in the context of equational properties E and we need to impose E-pre-regularity as follows. Given a kind-completed, pre-regular, order-sorted signature $\Sigma = (S, F, \leq)$ and given a set of equational axioms $E \in \mathbb{E}$, we assume throughout this section that all axioms for a symbol $f \in F$ are imposed at the kind level and apply to all overloaded versions of f. For example, $f : [\mathsf{s}] \times [\mathsf{s}] \rightarrow [\mathsf{s}]$ being commutative means that $f(x{:}[s], y{:}[s]) \doteq f(y{:}[s], x{:}[s])$ and this property is applicable to any term $f(t_1, t_2)$ such that $LS(t_1) = LS(t_2) = \mathsf{s}'$, and $\mathsf{s}' \leq [\mathsf{s}]$. For most constructions we then want to reason modulo E. This means that the relevant model is not the algebra of term $\mathcal{T}_\Sigma(\mathcal{X})$, but the quotient algebra $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ whose elements are

E-equivalence class $[t]_E$. Therefore, we need the least sort $LS(t)$ of a term t to be well-defined not only for terms $t \in \mathcal{T}_\Sigma(\mathcal{X})$, which pre-regularity ensures, but for the E-equivalence class to which t belongs. That is, we need $LS([t]_E)$ to be well-defined. Furthermore, we want $LS([t]_E)$ to be effectively determined by inspecting any representative $t' \in [t]_E$. If a pre-regular signature Σ satisfies these requirements for E, then we call it *E-pre-regular*.

A sufficient, checkable condition for E-pre-regularity can be obtained by dividing E into a disjoint union $E = E' \uplus U$, where E' consists only of C, A, and AC axioms, and U contains all the identity axioms. For Σ pre-regular, E'-pre-regularity can be ensured by proving that whenever $t =_{E'} t'$, then $LS(t) = LS(t')$. This, in turn, can be ensured by considering the finite set of downgrading substitutions ρ of each axiom $u \doteq v \in E'$ and checking that $LS(u\rho) = LS(v\rho)$ for each of them. For the axioms in U, it is enough to show that if $t =_U t'$, then $LS(t{\downarrow_U}) = LS(t'{\downarrow_U})$, where $t{\downarrow_U}$ denotes the canonical form obtained by applying the identity equations of the form $f(x{:}[\mathsf{s}], e) \doteq x{:}[\mathsf{s}]$ or $f(e, x{:}[\mathsf{s}]) \doteq x{:}[\mathsf{s}]$ in U as simplification rewrite rules oriented from left to right until no more rule application is possible and all occurrences of the identity symbols have disappeared. In this case, we also have to show that the equations in U are *sort-decreasing*, *i.e.,* for each downgrading $\rho = \{x{:}[\mathsf{s}] \mapsto x'{:}\mathsf{s}'\}$, we have $s' \leq LS(f(x'{:}\mathsf{s}', e))$ and $s' \leq LS(f(e, x'{:}\mathsf{s}'))$. If all these checks are satisfied, then it is easy to show that $\Sigma = (S, F, \leq)$ is E-pre-regular, with $LS([t]_E)$ computable as $LS(t{\downarrow_U})$, where $E = E' \uplus U$ as described above.

Indeed, checking E-pre-regularity is not only easy to do, but is *automated* in systems like Maude, as explained in (Clavel et al., 2007, 22.2.5). Our Maude-based implementation of order-sorted E-generalization described in this section takes order-sorted signatures Σ modulo axioms $E \in \mathbb{E}$ as inputs in the form of Maude functional or system modules, and relies on the *automatic* E-pre-regularity check performed by Maude for those modules as a necessary precondition for the correctness of the generalization algorithm.

As in Section 4, we consider two terms t and t' having the same top sort, otherwise they are incomparable and no generalizer exists. Starting from the initial configuration $\langle t \stackrel{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle$ where $[\mathsf{s}] = [LS(t)] = [LS(t')]$, configurations are transformed until a terminal configuration $\langle \emptyset \mid S \mid \theta \rangle$ is reached. Also, as in Section 5, when different function symbols satisfying different associativity and/or commutativity and/or identity axioms are considered, we can use the inference rules of Figures 14, 15, 16, 17, and 18 all together.

Note that we have just followed the same approach of Section 4 and extended the inference rules of Figures 6, 7, 8, 10, and 12 to Figures 14, 15, 16, 17, and 18 provided below. Note that sort declarations for binary function symbols that have either associativity, commutativity, or an identity element, have the same top sort for both arguments and for the result.

Termination is straightforward.

Theorem 26 (Termination). *Given a kind-completed, E-pre-regular, order-sorted equational theory (Σ, E) with $E \in \mathbb{E}$, terms t and t' such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, and a fresh variable $x{:}[\mathsf{s}]$, every derivation stemming from an*

$$\textbf{Decompose}_E \quad \frac{f \in (\Sigma \cup \mathcal{X}) \wedge ax(f) = \emptyset \wedge f : [\mathsf{s_1}] \times \ldots \times [\mathsf{s_n}] \to [\mathsf{s}]}{\begin{array}{c}\langle f(t_1, \ldots, t_n) \overset{x:[\mathsf{s}]}{\triangleq} f(t'_1, \ldots, t'_n) \wedge C \mid S \mid \theta\rangle \to \\ \langle t_1 \overset{x_1:[\mathsf{s_1}]}{\triangleq} t'_1 \wedge \ldots \wedge t_n \overset{x_n:[\mathsf{s_n}]}{\triangleq} t'_n \wedge C \mid S \mid \theta\sigma\rangle\end{array}}$$

where $\sigma = \{x{:}[\mathsf{s}] \mapsto f(x_1{:}[\mathsf{s_1}], \ldots, x_n{:}[\mathsf{s_n}])\}$, $x_1{:}[\mathsf{s_1}], \ldots, x_n{:}[\mathsf{s_n}]$ are fresh variables, and $n \geq 0$

$$\textbf{Solve}_E \quad \frac{\begin{array}{c}f = root(t) \wedge g = root(t') \wedge f \neq g \wedge U_f \notin ax(f) \wedge U_g \notin ax(g) \wedge \\ \wedge s' \in LUBS(LS(t), LS(t')) \wedge \nexists y \nexists \mathsf{s''} : t \overset{y:\mathsf{s''}}{\triangleq} t' \in^E S\end{array}}{\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \wedge C \mid S \mid \theta\rangle \to \langle C \mid S \wedge t \overset{z:\mathsf{s'}}{\triangleq} t' \mid \theta\rangle}$$

where $\sigma = \{x{:}[\mathsf{s}] \mapsto z{:}\mathsf{s'}\}$ and $z{:}\mathsf{s'}$ is a fresh variable.

$$\textbf{Recover}_E \quad \frac{root(t) \neq root(t') \wedge \exists y \exists \mathsf{s'} : t \overset{y:\mathsf{s'}}{\triangleq} t' \in^E S \wedge s'' \in LUBS(LS(t), LS(t'))}{\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \wedge C \mid S \mid \theta\rangle \to \langle C \mid S \mid \theta\sigma\rangle}$$

where $\sigma = \{x{:}[\mathsf{s}] \mapsto y{:}\mathsf{s''}\}$

Figure 14: Order-sorted basic inference rules for least general E–generalization

initial configuration $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle$ *using the inference rules of Figures 14, 15, 16, 17, and 18 terminates in a final configuration of the form* $\langle \emptyset \mid S \mid \theta\rangle$.

Proof. Similar to the proofs of Theorems 1 and 20. □

In order to prove correctness and completeness, Definitions 11, 14, 17, and 20 for E-conflict pairs are extended to the order-sorted case in the obvious way; recall that variables with the same name but different sorts, *e.g.,* $x{:}\mathsf{A}$ and $x{:}\mathsf{B}$, are considered as different variables.

We follow the same proof schema of Section 4.2 and define order-sorted E-lgg computation by subsort specialization. That is, to compute generalizers by removing sorts (*i.e.,* upgrading variables to top sorts), first, then computing (unsorted) E-lggs, and finally obtaining the right subsorts by a suitable post-processing. This approach is not used in practice, it is used only for the proofs of correctness and completeness of the inference rules.

First, for generalization in the modulo case, we introduce a special notation for subterm replacement when we have associative or associative-commutative conflict pairs. Note that E-pre-regularity is essential here because it ensures that after replacing a subterm by a variable, the least sort does not depend on the chosen representation within the equivalence class of a term, i.e., it does not depend on how the flattened version of the term is obtained.

Definition 21 (A-Subterm Replacement). *Given two flattened terms* t *and* t' *and an associative conflict pair* (u, v) *with a pair of conflict positions* $p \in$

Decompose$_C$

$$\frac{f : [\mathsf{s}] \times [\mathsf{s}] \to [\mathsf{s}] \wedge C_f \in ax(f) \wedge A_f \notin ax(f) \wedge i \in \{1,2\}}{\langle f(t_1,t_2) \overset{x:[\mathsf{s}]}{\triangleq} f(t'_1,t'_2) \wedge C \mid S \mid \theta\rangle \to \langle t_1 \overset{x_1:[\mathsf{s}]}{\triangleq} t'_i \wedge t_2 \overset{x_2:[\mathsf{s}]}{\triangleq} t'_{(i \bmod 2)+1} \wedge C \mid S \mid \theta\sigma\rangle}$$

where $\sigma = \{x{:}[\mathsf{s}] \mapsto f(x_1{:}[\mathsf{s}], x_2{:}[\mathsf{s}])\}$, and $x_1{:}[\mathsf{s}], x_2{:}[\mathsf{s}]$ are fresh variables

Figure 15: Order-sorted decomposition rule for a commutative function symbol f

Decompose$_{A-\text{left}}$

$$\frac{f : [\mathsf{s}] \times [\mathsf{s}] \to [\mathsf{s}] \wedge A_f \in ax(f) \wedge C_f \notin ax(f) \wedge n \geq 2 \wedge m \geq 2 \wedge k \in \{1,\ldots,n-1\}}{\langle f(t_1,\ldots,t_n) \overset{x:[\mathsf{s}]}{\triangleq} f(t'_1,\ldots,t'_m) \wedge C \mid S \mid \theta\rangle \to \langle f(t_1,\ldots,t_k) \overset{x_1:[\mathsf{s}]}{\triangleq} t'_1 \wedge f(t_{k+1},\ldots,t_n) \overset{x_2:[\mathsf{s}]}{\triangleq} f(t'_2,\ldots,t'_m) \wedge C \mid S \mid \theta\sigma\rangle}$$

where $\sigma = \{x{:}[\mathsf{s}] \mapsto f(x_1{:}[\mathsf{s}], x_2{:}[\mathsf{s}])\}$, and $x_1{:}[\mathsf{s}], x_2{:}[\mathsf{s}]$ are fresh variables

Decompose$_{A-\text{right}}$

$$\frac{f : [\mathsf{s}] \times [\mathsf{s}] \to [\mathsf{s}] \wedge A_f \in ax(f) \wedge C_f \notin ax(f) \wedge n \geq 2 \wedge m \geq 2 \wedge k \in \{1,\ldots,m-1\}}{\langle f(t_1,\ldots,t_n) \overset{x:[\mathsf{s}]}{\triangleq} f(t'_1,\ldots,t'_m) \wedge C \mid S \mid \theta\rangle \to \langle t_1 \overset{x_1:[\mathsf{s}]}{\triangleq} f(t'_1,\ldots,t'_k) \wedge f(t_2,\ldots,t_n) \overset{x_2:[\mathsf{s}]}{\triangleq} f(t'_{k+1},\ldots,t'_m) \wedge C \mid S \mid \theta\sigma\rangle}$$

where $\sigma = \{x{:}[\mathsf{s}] \mapsto f(x_1{:}[\mathsf{s}], x_2{:}[\mathsf{s}])\}$, and $x_1{:}[\mathsf{s}], x_2{:}[\mathsf{s}]$ are fresh variables

Figure 16: Order-sorted decomposition rules for an associative (non–commutative) function symbol f

$Pos(t)$ and $p' \in Pos(t')$ such that $u = f(u_1,\ldots,u_m)$, $m \geq 1$, $v = f(v_1,\ldots,v_n)$, $n \geq 1$, f is associative, $t|_p = f(w_1,\ldots,w_{k_1},u_1,\ldots,u_m,w'_1,\ldots,w'_{k_2})$, $k_1 \geq 0$, $k_2 \geq 0$, and $t'|_{p'} = f(w''_1,\ldots,w''_{k_3},v_1,\ldots,v_n,w'''_1,\ldots,w'''_{k_4})$, $k_3 \geq 0$, $k_4 \geq 0$, we write $t[[x{:}\mathsf{s}]]_p$ and $t'[[x{:}\mathsf{s}]]_{p'}$ to denote the terms $t[[x{:}\mathsf{s}]]_p = t[f(w_1,\ldots,w_{k_1},x{:}\mathsf{s},w'_1,\ldots,w'_{k_2})]_p$, and $t'[[x{:}\mathsf{s}]]_{p'} = t[f(w''_1,\ldots,w''_{k_3},x{:}\mathsf{s},w'''_1,\ldots,w'''_{k_4})]_{p'}$.

Definition 22 (AC-Subterm Replacement). *Given two flattened terms t and t' and an associative-commutative conflict pair (u,v) with a pair of conflict positions $p \in Pos(t)$ and $p' \in Pos(t')$ such that $u = f(u_1,\ldots,u_m)$, $m \geq 1$, $v = f(v_1,\ldots,v_n)$, $n \geq 1$, f is associative-commutative, $t|_p = f(w_1,\ldots,w_{k_1})$ s.t. for each $i \in \{1,\ldots,m\}$, there is $j \in \{1,\ldots,k_1\}$ with $u_i =_E w_j$, and $t'|_{p'} = f(w'_1,\ldots,w'_{k_2})$ s.t. for each $i \in \{1,\ldots,n\}$, there is $j \in \{1,\ldots,k_2\}$ with $v_i =_E w'_j$, then we write $t[[x{:}\mathsf{s}]]_p$ and $t'[[x{:}\mathsf{s}]]_{p'}$ to denote the terms $t[[x{:}\mathsf{s}]]_p =$*

Decompose$_{AC-\text{left}}$

$$\frac{f : [\mathsf{s}] \times [\mathsf{s}] \to [\mathsf{s}] \wedge \{A_f, C_f\} \subseteq ax(f) \wedge n \geq 2 \wedge m \geq 2 \wedge \\ \{i_1, \ldots, i_n\} = \uplus\{1, \ldots, n\} \ \wedge k_n \in \{1, \ldots, n-1\} \wedge k_m \in \{1, \ldots, m\}}{\begin{array}{c} \langle f(t_1, \ldots, t_n) \overset{x:[\mathsf{s}]}{\triangleq} f(t'_1, \ldots, t'_m) \wedge C \mid S \mid \theta \rangle \to \\ \langle f(t_{i_1}, \ldots, t_{i_{k_n}}) \overset{x_1:[\mathsf{s}]}{\triangleq} t'_{k_m} \wedge \\ f(t_{i_{(k_n+1)}}, \ldots, t_{i_n}) \overset{x_2:[\mathsf{s}]}{\triangleq} f(t'_1, \ldots, t'_{k_m-1}, t'_{k_m+1}, \ldots, t'_m) \wedge C \mid S \mid \theta\sigma \rangle \end{array}}$$

where $\sigma = \{x{:}[\mathsf{s}] \mapsto f(x_1{:}[\mathsf{s}], x_2{:}[\mathsf{s}])\}$, and $x_1{:}[\mathsf{s}], x_2{:}[\mathsf{s}]$ are fresh variables

Decompose$_{AC-\text{right}}$

$$\frac{f : [\mathsf{s}] \times [\mathsf{s}] \to [\mathsf{s}] \wedge \{A_f, C_f\} \subseteq ax(f) \wedge n \geq 2 \wedge m \geq 2 \wedge \\ \{i_1, \ldots, i_m\} = \uplus\{1, \ldots, m\} \ \wedge k_m \in \{1, \ldots, m-1\} \wedge k_n \in \{1, \ldots, n\}}{\begin{array}{c} \langle f(t_1, \ldots, t_n) \overset{x:[\mathsf{s}]}{\triangleq} f(t'_1, \ldots, t'_m) \wedge C \mid S \mid \theta \rangle \to \\ \langle t_{k_n} \overset{x_1:[\mathsf{s}]}{\triangleq} f(t'_{i_1}, \ldots, t'_{i_{k_m}}) \wedge \\ f(t_1, \ldots, t_{k_n-1}, t_{k_n+1}, \ldots, t_n) \overset{x_2:[\mathsf{s}]}{\triangleq} f(t'_{i_{(k_m+1)}}, \ldots, t'_{i_m}) \wedge C \mid S \mid \theta\sigma \rangle \end{array}}$$

where $\sigma = \{x{:}[\mathsf{s}] \mapsto f(x_1{:}[\mathsf{s}], x_2{:}[\mathsf{s}])\}$, and $x_1{:}[\mathsf{s}], x_2{:}[\mathsf{s}]$ are fresh variables

Figure 17: Order-sorted decomposition rules for an associative–commutative function symbol f

$t[f(\overline{w_1}, \ldots, \overline{w_{k_1}}, x{:}\mathsf{s})]_p$ *where* $\{\overline{w_1}, \ldots, \overline{w_{k_1}}\} = \bigcup\{w_i \mid 1 \leq i \leq k_1, \forall 1 \leq j \leq n, w_i \neq_E u_j\}$, *and* $t'[[x{:}\mathsf{s}]]_{p'} = t[f(\overline{w'_1}, \ldots, \overline{w'_{k_2}}, x{:}\mathsf{s})]_{p'}$ *where* $\{\overline{w'_1}, \ldots, \overline{w'_{k_2}}\} = \bigcup\{w'_i \mid 1 \leq i \leq k_2, \forall 1 \leq j \leq n, w'_i \neq_E v_j\}$.

As in Section 4.2, we define order-sorted E-lgg computation by subsort specialization using a top-sorted generalization (see Definition 6) and a sort-specialized generalization (see Definition 24).

Definition 23 (Top-sorted Equational Generalization). *Given a kind-completed, E-pre-regular, order-sorted equational theory (Σ, E) with $E \in \mathbb{E}$, and flattened Σ-terms t and t' such that $[LS(t)] = [LS(t')]$, let $(u_1, v_1), \ldots, (u_k, v_k)$ be the E-conflict pairs of t and t', and for each such conflict pair (u_i, v_i), let $(p^i_1, \ldots, p^i_{n_i}, q^i_1, \ldots, q^i_{n_i})$, $1 \leq i \leq k$, be the corresponding E-conflict positions, and let $[\mathsf{s_i}] = [LS(u_i)] = [LS(v_i)]$. We define the term denoting the* top order-sorted equational least general generalization *as*

$$tsg_E(t, t') = t[[x^1_1{:}[\mathsf{s_1}], \ldots, x^1_{n_1}{:}[\mathsf{s_1}]]]_{p^1_1, \ldots, p^1_{n_1}} \cdots [[x^k_1{:}[\mathsf{s_k}], \ldots, x^k_{n_k}{:}[\mathsf{s_k}]]]_{p^k_1, \ldots, p^k_{n_k}}$$

where $x^1_1{:}[\mathsf{s_1}], \ldots, x^1_{n_1}{:}[\mathsf{s_1}], \ldots, x^k_1{:}[\mathsf{s_k}], \ldots, x^k_{n_k}{:}[\mathsf{s_k}]$ are fresh variables.

The order-sorted equational lgg's are obtained by subsort specialization.

$$\textbf{Expand}_U \ \frac{\begin{array}{c} f : [\mathsf{s}] \times [\mathsf{s}] \to [\mathsf{s}] \wedge U_f \in ax(f) \wedge root(t) \equiv f \wedge t' \neq_E e \wedge \\ root(t') \neq f \wedge t'' \in \{f(e,t'), f(t',e)\} \end{array}}{\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \wedge C \mid S \mid \theta \rangle \to \langle t \overset{x:[\mathsf{s}]}{\triangleq} t'' \wedge C \mid S \mid \theta \rangle}$$

Figure 18: Order-sorted inference rule for expanding function symbol f with identity element e

Definition 24 (Sort-specialized Equational Generalization). *Given a kind-completed, E-pre-regular, order-sorted equational theory (Σ, E) with $E \in \mathbb{E}$, and flattened Σ-terms t and t' such that $[LS(t)] = [LS(t')]$, let (u_1, v_1), $\ldots, (u_k, v_k)$ be the conflict pairs of t and t', and for each such conflict pair (u_i, v_i), let $p^i_1, \ldots, p^i_{n_i}$, $1 \leq i \leq k$, be the corresponding E-conflict positions, let $[\mathsf{s_i}] = [LS(u_i)] = [LS(v_i)]$, and let $x^1_1{:}[\mathsf{s_1}], \ldots, x^1_{n_1}{:}[\mathsf{s_1}], \ldots, x^k_1{:}[\mathsf{s_k}], \ldots, x^k_{n_k}{:}[\mathsf{s_k}]$ be the variable identifiers used in Definition 23. We define*

$$\begin{array}{rl} \textit{sort-down-subs}_E(t,t') = \{\rho \mid & Dom(\rho) = \{x^1_1{:}[\mathsf{s_1}], \ldots, x^1_{n_1}{:}[\mathsf{s_1}], \ldots, x^k_1{:}[\mathsf{s_k}], \ldots, x^k_{n_k}{:}[\mathsf{s_k}]\} \wedge \\ & \forall 1 \leq i \leq k, \forall 1 \leq j \leq n_i : \\ & (x^i_j{:}[\mathsf{s_i}])\rho = x_i{:}\mathsf{s}'_\mathsf{i} \wedge \mathsf{s}'_\mathsf{i} \in LUBS(LS(u_i), LS(v_i))\} \end{array}$$

where all the $x_i{:}\mathsf{s}'_\mathsf{i}$ are fresh variables. The set of sort-specialized E-generalizers is defined as $ssg_E(t,t') = \{tsg_E(t,t')\rho \mid \rho \in \textit{sort-down-subs}_E(t,t')\}$.

Now, we prove that sort-specialized E-generalizations are the same as order-sorted E-lggs.

Theorem 27. *Given a kind-completed, E-pre-regular, order-sorted equational theory (Σ, E) with $E \in \mathbb{E}$, and flattened Σ-terms t and t' such that $[LS(t)] = [LS(t')]$, $tsg_E(t,t')$ is an order-sorted equational generalizer of t and t', and $lgg_E(t,t')$ provides a minimal complete set of order-sorted equational lggs.*

Proof. Similar to the proof of Theorem 6. □

Finally, we prove the correctness and completeness of the order-sorted, equational generalization algorithm.

Theorem 28 (Correctness). *Given a kind-completed, E-pre-regular, order-sorted equational theory (Σ, E) with $E \in \mathbb{E}$, flattened Σ-terms t and t' such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, and a fresh variable $x{:}[\mathsf{s}]$, if $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle \to^* \langle \emptyset \mid S \mid \theta \rangle$ using the inference rules of Figures 14, 15, 16, 17, and 18, then $(x{:}[\mathsf{s}])\theta$ is a generalizer of t and t'.*

Theorem 29 (Completeness). *Given a kind-completed, E-pre-regular, order-sorted equational theory (Σ, E) with $E \in \mathbb{E}$, flattened Σ-terms t and t' such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, and a fresh variable $x{:}[\mathsf{s}]$, if u is a least general generalizer of t and t' modulo E, then there is a derivation $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle \to^* \langle C \mid S \mid \theta \rangle$ using the inference rules of Figures 14, 15, 16, 17, and 18 such that $u \equiv_E (x{:}[\mathsf{s}])\theta$.*

7. Implementation

We have implemented the different calculi proposed in this paper in the ACUOS system, which is publicly available[1]. The tool is written in the high-performance, rewriting logic language Maude (Clavel et al., 2007), whose powerful reflective capabilities allow a straightforward translation of the inference systems presented in this paper into less than 1000 lines of code. To the best of our knowledge, ACUOS is the first generalization system that computes least general generalizers in order-sorted theories modulo equational axioms.

Roughly speaking, ACUOS proceeds as follows. First, we lift to the meta-level the two input terms and the module that contains the considered order-sorted theory[2]; the three of them are passed as arguments to ACUOS which builds the initial configuration. Then the implementation performs an exhaustive search using Maude's `metaSearch` function, which executes the inference rules in the calculi described above as meta-level rewrite rules in order to obtain a complete set of least general generalizers. Since the computed set is generally not minimal, a final filtering step is performed that allows us to get rid of those elements that do not satisfy the maximality property given in Definition 9. This filtering is performed by pairwise comparison of all elements in the set using the ordering $\leq_E$, discarding any term u that subsumes *modulo* E any other term v in the set, *i.e.,* $u <_E v$. We use Maude's `metaMatch` for this task, since it provides a simple and efficient means to check the relation $\leq_E$. If $u \equiv_E v$ (*i.e.,* $u \leq_E v$ and $v \leq_E u$), the terms u and v belong to the same equivalence class and hence the choice of the class representative is irrelevant; we arbitrarily choose a representative based on the size of the term.

ACUOS can be accessed in three useful ways that provide the same functionality: using a Maude routine, a Full Maude user-level command, or the ACUOS Web interface. The Maude routine is a direct invocation to the backend of the tool. The Full Maude user-level command allows the user to harness the power of ACUOS while being relieved from most technicalities. Finally, the ACUOS Web interface combines AJAX and Java technologies to allow the tool to be used through any Web browser.

Let us illustrate the use of the Full Maude command with an example. Consider the following Full Maude module (we refer the reader to (Clavel et al., 2007) for Maude and Full Maude syntax):

```
fmod SPECIFICATION is
   sort A B C D E .
   subsort A B < C D < E .

   op a : -> A .
   op b : -> B .
   op c : -> C .
```

[1] At `http://safe-tools.dsic.upv.es/acuos/`

[2] More precisely, the module can be either an equational theory of the form `fmod` $(\Sigma, E \cup G)$ `endf` with $E \in \mathbb{E}$ (a functional module), or a rewrite theory of the form `mod` $(\Sigma, E \cup G, R)$ `endm` with $E \in \mathbb{E}$ (a system module).

```
  op d : -> D .
  op e : -> E .
  op none : -> E .

  op f : A A -> A [assoc comm id: none] .
  op f : B B -> B [assoc comm id: none] .
  op f : C C -> C [assoc comm id: none] .
  op f : D D -> D [assoc comm id: none] .
  op f : E E -> E [assoc comm id: none] .
endfm
```

This module is automatically extended to its kind-complete version by Maude. It defines five constants `a`, `b`, `c`, `d`, `e` and five binary symbols sharing the same name `f` but with different signatures corresponding to the subsort structure of Figure 1. All five versions of symbol f (plus its kind extension $f : [\mathsf{E}] \to [\mathsf{E}]$) are associative-commutative and have the constant `none` as their identity element. Now, we can type the following generalization problem in Full Maude obtaining the six possible order-sorted E-lggs.

```
(lggs in fACU-OS : f(b,b,a) =? f(a,a,b) .)

Solutions:
f(#0:C, #0:C, #1:C)
f(#0:C, #0:C, #1:D)
f(#0:C, #1:D, #1:D)
f(#0:D, #0:D, #1:D)
f(a, b, #0:C)
f(a, b, #0:D)
```

Figure 19 and Figure 20 respectively show the equivalent input form and results window for the previous example, using the ACUOS Web interface.

ACUOS is applicable to any functional or system Maude module, including the standard modules provided in the Maude distribution. The following example shows the application of ACUOS to two Peano arithmetic expressions, as defined in the Maude Prelude module.

```
(lggs in NAT :
    s(s(s(X:Nat))) + s(s(s(X:Nat))) + s(s(s(s(s(X:Nat)))))
    =?
    s(s(s(X:Nat))) + s(s(s(s(s(X:Nat))))) + s(s(s(s(s(X:Nat))))) .)

Solutions:
s(s(s(#0:Nat))) + s(s(s(#0:Nat))) + s(s(s(#1:Nat)))
s(s(s(#0:Nat))) + s(s(s(#1:Nat))) + s(s(s(s(s(#0:Nat)))))
```

This feature is particularly interesting because it enables ACUOS to reason about any Maude entity that has a meta-representation, such as modules, operator declarations, equation and rule definitions, *etc.* For example, we are able to generalize operator declarations in Full Maude as follows:

```
(lggs in META-LEVEL :
 (op g : A B -> B [assoc comm] .) =? (op f : A C -> C [assoc] .) .)

Solutions:
op #0:Qid : A #1:Qid -> #1:Qid [ #1:AttrSet assoc ] .
```

ACUOS Online Tool

Select model: fACU OS ▼ Load

Module:

```
fmod SPECIFICATION is

   sort A B C D E .
   subsort A B < C D < E .

   op f : A A -> A [assoc comm id: none] .
   op f : B B -> B [assoc comm id: none] .
   op f : C C -> C [assoc comm id: none] .
   op f : D D -> D [assoc comm id: none] .
   op f : E E -> E [assoc comm id: none] .

   op a : -> A .
   op b : -> B .
   op c : -> C .
   op d : -> D .
   op e : -> E .
   op none : -> E .

endfm
```

Term 1:

f(a,a,b)

Term 2:

f(b,b,a)

Generalize!

Figure 19: Input form of the Web interface of the ACUOS tool

Generalization results

Generalizers

1. f(#0:C, #0:C, #2:C)
2. f(#0:C, #0:C, #3:D)
3. f(#0:C, #3:D, #3:D)
4. f(#0:C, a, b)
5. f(#1:D, #1:D, #3:D)
6. f(#1:D, a, b)

Figure 20: Output window of the Web interface of the ACUOS tool

8. Conclusion and Future Work

We have presented an order-sorted, modular equational generalization algorithm that computes a minimal and complete set of least general generalizers 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 many-sorted and order-sorted declarative language and equational reasoning system (and also, a fortiori, to untyped languages and systems which have only one sort). As shown in the examples, the algorithms we propose are effective to compute E-generalizers, which would be unfeasible in a naïve way.

In our own work, we plan to use the proposed 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.,* (Albert et al., 1999; Alpuente et al., 1998a,b, 1999). We are also considering adding this generalization mechanism to an inductive theorem prover such a Maude's ITP (Clavel and Palomino, 2005) 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 (Boyer and Moore, 1980) and its ACL2 successor (Kaufmann et al., 2000).

Acknowledgements

We gratefully acknowledge Pedro Ojeda for a preliminary implementation of the order-sorted and equational generalization algorithms. We are also thankful to Hassan Aït-Kaci and Temur Kutsia for useful remarks on (Alpuente et al., 2009a,b). Finally, we thank the anonymous referees for useful insights on order-sorted generalization.

References

Aït-Kaci, H., 1983. Outline of a calculus of type subsumption. Tech. Rep. MS-CIS-83-34, C.I.S. Department, University of Pennsylvania. August 1983.

Aït-Kaci, H., 2007. Data models as constraint systems: A key to the semantic web. Constraint Programming Letters 1, 33–88.

Aït-Kaci, H., Sasaki, Y., 2001. An axiomatic approach to feature term generalization. In: Raedt, L. D., Flach, P. A. (Eds.), Machine Learning: EMCL 2001, 12th European Conference on Machine Learning, Freiburg, Germany, September 5-7, 2001, Proceedings. Vol. 2167 of Lecture Notes in Computer Science. Springer, pp. 1–12.

Albert, E., Alpuente, M., Hanus, M., Vidal, G., 1999. A partial evaluation framework for curry programs. In: Ganzinger, H., McAllester, D. A., Voronkov, A. (Eds.), Logic Programming and Automated Reasoning, 6th

International Conference, LPAR'99, Tbilisi, Georgia, September 6-10, 1999, Proceedings. Vol. 1705 of Lecture Notes in Computer Science. Springer, pp. 376–395.

Alpuente, M., Escobar, S., Meseguer, J., Ojeda, P., 2009a. A Modular Equational Generalization Algorithm. In: Proc. 18th Int'l Symp. on Logic-Based Program Synthesis and Transformation, LOPSTR 2008, Revised Selected Papers. Vol. 5438 of Lecture Notes in Computer Science. Springer, pp. 24–39.

Alpuente, M., Escobar, S., Meseguer, J., Ojeda, P., 2009b. Order–Sorted Generalization. Electr. Notes Theor. Comput. Sci. 246, 27–38.

Alpuente, M., Falaschi, M., Levi, G., 1995. Incremental Constraint Satisfaction for Equational Logic Programming. Theoretical Computer Science 142 (1), 27–57.

Alpuente, M., Falaschi, M., Vidal, G., 1998a. Partial evaluation of functional logic programs. ACM Trans. Program. Lang. Syst. 20 (4), 768–844.

Alpuente, M., Falaschi, M., Vidal, G., 1998b. A unifying view of functional and logic program specialization. ACM Comput. Surv. 30 (3es), 9.

Alpuente, M., Hanus, M., Lucas, S., Vidal, G., 1999. Specialization of Inductively Sequential Functional Logic Programs. In: Proc. 4th ACM SIGPLAN International Conference on Functional Programming (ICFP '99), Paris, France, September 27-29, 1999. pp. 273–283.

Armengol, E., 1998. A framework for integrating learning and problem solving. Ph.D. thesis, Institut d'Investigació en Intel.ligència Artificial (IIA).

Armengol, E., Plaza, E., 2000. Bottom-Up Induction of Feature Terms. Machine Learning 41 (3), 259–294.

Baader, F., 1991. Unification, weak unification, upper bound, lower bound, and generalization problems. In: Book, R. V. (Ed.), Rewriting Techniques and Applications, 4th International Conference, RTA-91, Como, Italy, April 10-12, 1991, Proceedings. Vol. 488 of Lecture Notes in Computer Science. Springer, pp. 86–97.

Baader, F., Snyder, W., 1999. Unification theory. In: Handbook of Automated Reasoning. Elsevier.

Belli, F., Jack, O., 1998. Declarative paradigm of test coverage. Softw. Test., Verif. Reliab. 8 (1), 15–47.

Bergstra, J., Heering, J., Klint, P., 1989. Algebraic Specification. ACM Press.

Borovanský, P., Kirchner, C., Kirchner, H., Moreau, P.-E., 2002. ELAN from a rewriting logic point of view. Theoretical Computer Science 285, 155–185.

Boyer, R., Moore, J., 1980. A Computational Logic. Academic Press.

Bulychev, P. E., Kostylev, E. V., Zakharov, V. A., 2010. Anti-unification algorithms and their applications in program analysis.

Burghardt, J., 2005. E-generalization using Grammars. Artif. Intell. 165 (1), 1–35.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C., 2007. All About Maude - A High-Performance Logical Framework. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Clavel, M., Palomino, M., 2005. The ITP tool's manual, universidad Complutense, Madrid, `http://maude.sip.ucm.es/itp/`.

Diaconescu, R., Futatsugi, K., 1998. CafeOBJ Report. Vol. 6 of AMAST Series in Computing. World Scientific, AMAST Series.

Eker, S., 2003. Associative-commutative rewriting on large terms. In: Nieuwenhuis, R. (Ed.), Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings. Vol. 2706 of Lecture Notes in Computer Science. Springer, pp. 14–29.

Frisch, A. M., Page, C. D. J., 1990. Generalization with taxonomic information. In: AAAI. pp. 755–761.

Gallagher, J. P., 1993. Tutorial on specialisation of logic programs. In: PEPM '93: Proceedings of the 1993 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation. ACM, New York, NY, USA, pp. 88–98.

Goguen, J., Meseguer, J., 1992. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theoretical Computer Science 105, 217–273.

Goguen, J., Winkler, T., Meseguer, J., Futatsugi, K., Jouannaud, J.-P., 2000. Introducing OBJ. In: Software Engineering with OBJ: Algebraic Specification in Action. Kluwer, pp. 3–167.

Huet, G., 1976. Résolution d'équations dans des langages d'ordre 1, 2,...,ω. Thèse de doctorat d'état en sciences mathématiques, Univ. Paris VII.

Hullot, J.-M., 1980. Canonical Forms and Unification. In: 5th Int'l Conference on Automated Deduction CADE'80. Vol. 87 of LNCS. Springer-Verlag, Berlin, pp. 318–334.

Jouannaud, J.-P., Kirchner, C., 1991. Solving Equations in Abstract Algebras: A Rule-Based Survey of Unification. In: Computational Logic - Essays in Honor of Alan Robinson. MIT Press, pp. 257–321.

Kaufmann, M., Manolios, P., Moore, J., 2000. Computer-Aided Reasoning: An Approach. Kluwer.

Kitzelmann, E., Schmid, U., 2006. Inductive synthesis of functional programs: An explanation based generalization approach. Journal of Machine Learning Research 7, 429–454.

Kutsia, T., Levy, J., Villaret, M., 2011. Anti-unification for unranked terms and hedges. In: Schmidt-Schauß, M. (Ed.), Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia. Vol. 10 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 219–234.

Lassez, J.-L., Maher, M. J., Marriott, K., 1988. Unification Revisited. In: Minker, J. (Ed.), Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, Los Altos, Ca., pp. 587–625.

Lu, J., Mylopoulos, J., Harao, M., Hagiya, M., 2000. Higher order generalization and its application in program verification. Ann. Math. Artif. Intell. 28 (1-4), 107–126.

Martelli, A., Montanari, U., 1982. An efficient unification algorithm. Transactions on Programming Languages and Systems 4 (2), 258–282.

Meseguer, J., 1998. Membership algebra as a logical framework for equational specification. In: Parisi-Presicce, F. (Ed.), Proc. WADT'97. Springer LNCS 1376, pp. 18–61.

Meseguer, J., Goguen, J., Smolka, G., 1989. Order-sorted unification. J. Symbolic Computation 8, 383–413.

Mogensen, T. Æ., 2000. Glossary for partial evaluation and related topics. Higher-Order and Symbolic Computation 13 (4).

Muggleton, S., 1999. Inductive Logic Programming: Issues, Results and the Challenge of Learning Language in Logic. Artif. Intell. 114 (1-2), 283–296.

Østvold, B., 2004. A functional reconstruction of anti-unification. Tech. Rep. DART/04/04, Norwegian Computing Center.

Pfenning, F., 1991. Unification and anti-unification in the calculus of constructions. In: Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science, 15-18 July, 1991, Amsterdam, The Netherlands. IEEE Computer Society, pp. 74–85.

Plaza, E., 1995. Cases as terms: A feature term approach to the structured representation of cases. In: Veloso, M. M., Aamodt, A. (Eds.), Case-Based Reasoning Research and Development, First International Conference, ICCBR-95, Sesimbra, Portugal, October 23-26, 1995, Proceedings. Vol. 1010 of Lecture Notes in Computer Science. Springer, pp. 265–276.

Plotkin, G., 1970. A note on inductive generalization. In: Machine Intelligence. Vol. 5. Edinburgh University Press, pp. 153–163.

Plotkin, G., 2004. A structural approach to operational semantics. J. Log. Algebr. Program. 60-61, 17–139.

Popplestone, R., 1969. An experiment in automatic induction. In: Machine Intelligence. Vol. 5. Edinburgh University Press, pp. 203–215.

Pottier, L., 1989. Generalisation de termes en theorie equationelle: Cas associatif-commutatif. Tech. Rep. INRIA 1056, Norwegian Computing Center.

Reynolds, J., 1970. Transformational systems and the algebraic structure of atomic formulas. Machine Intelligence 5, 135–151.

Schmidt-Schauss, M., 1986. Unification in many-sorted equational theories. In: Proceedings, 8th International Conference on Automated Deduction. Springer-Verlag, pp. 538–552, lNCS, Volume 230.

Siekmann, J., 1989. Unification Theory. Journal of Symbolic Computation 7, 207–274.

Smolka, G., Aït-Kaci, H., 1989. Inheritance hierarchies: Semantics and unification. J. Symb. Comput. 7 (3/4), 343–370.

Smolka, G., Nutt, W., Goguen, J., Meseguer, J., 1989. Order-sorted equational computation. In: Nivat, M., Aït-Kaci, H. (Eds.), Resolution of Equations in Algebraic Structures. Vol. 2. Academic Press, pp. 297–367.

TeReSe (Ed.), 2003. Term Rewriting Systems. Cambridge University Press, Cambridge.