UNIVERSIDAD POLITECNICA DE VALENCIA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

# ACUOS
# A System for Order-Sorted Modular ACU Generalization

**Master's Thesis**

**Presented by:**
Javier Espert Real

**Supervisors:**
María Alpuente Frasnedo
Santiago Escobar Román

July 9th, 2012

# Abstract

Generalization, also called anti-unification, is the dual of unification. Given terms $t$ and $t'$, a generalization 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 generalization (lgg). In this thesis, 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 set of lggs, so that any other generalization 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, data mining, and theorem proving for typed equational reasoning systems and typed rule-based languages such as ASF+SDF, Elan, OBJ, Cafe-OBJ, and Maude.

This thesis also describes a tool for automatically computing the generalizers of a given set of structures in a typed language *modulo* a set of axioms. By supporting the modular combination of associative, commutative and unity (ACU) equational attributes for arbitrary function symbols, modular ACU generalization adds enough expressive power to ordinary generalization to reason about typed data structures such as lists, sets and multisets. The ACU generalization technique has been generally and efficiently implemented in the ACUOS system and can be easily integrated with third-party software.

**Keywords:** least general generalization, rule–based languages, equational reasoning

# Resumen

La generalización, también denominada anti-unificación, es la operación dual de la unificación. Dados dos términos $t$ y $t'$, un generalizador es un término $t''$ del cual $t$ y $t'$ son instancias de sustitución. El concepto dual del unificador más general (mgu) es el de generalizador menos general (lgg). En esta tesina extendemos el conocido algoritmo de generalización sin tipos a, primero, una configuración *order-sorted* con *sorts*, *subsorts* y *polimorfismo de subtipado*; en segundo lugar, la extendemos para soportar generalización módulo teorías ecuacionales, donde los símbolos de función pueden obedecer cualquier combinación de axiomas de asociatividad, conmutatividad e identidad (incluyendo el conjunto vacío de dichos axiomas); y, en tercer lugar, a la combinación de ambos, que resulta en un algoritmo modular de generalización *order-sorted* ecuacional. A diferencia de las configuraciones sin tipos, en nuestro marco teórico en general el lgg no es único, que se debe tanto al tipado como a los axiomas ecuacionales. En su lugar, existe un conjunto finito y mínimo de lggs, tales que cualquier otra generalización tiene a alguno de ellos como instancia. Nuestros algoritmos de generalización se expresan mediante reglas de inferencia para las cuales damos demostraciones de corrección. Ello abre la puerta a nuevas aplicaciones en campos como la evaluación parcial, la síntesis de programas, la minería de datos y la demostración de teoremas para sistemas de razonamiento ecuacional y lenguajes tipados basados en reglas tales como ASD+SDF, Elan, OBJ, CafeOBJ y Maude.

Esta tesis también describe una herramienta para el cómputo automatizado de los generalizadores de un conjunto dado de estructuras en un lenguaje tipado *módulo* un conjunto de axiomas dado. Al soportar la combinación modular de atributos ecuacionales de asociatividad, conmutatividad y existencia de elemento neutro (ACU) para símbolos de función arbitrarios, la generalización ACU modular aporta suficiente poder expresivo a la generalización ordinaria para razonar sobre estructuras de datos tipadas tales como listas, conjuntos y multiconjuntos. La técnica ha sido implementada con generalidad y eficiencia en el sistema ACUOS y puede ser fácilmente integrada con software de terceros.

**Palabras clave:** generalización menos general, lenguajes basados en reglas, razonamiento ecuacional

# Contents

# List of Figures

# Introduction

Generalization is a formal reasoning component of many symbolic frameworks, including theorem provers, and automatic program analysis, synthesis, verification, compilation, refactoring, test case generation, learning, specialisation, and transformation techniques see, e.g., (Gallagher, 1993; Muggleton, 1999; Boyer and Moore, 1980a; Pfenning, 1991; Lu et al., 2000; Kitzelmann and Schmid, 2006; Bulychev et al., 2010; Kutsia et al., 2011). Generalization, also called anti-unification, is the dual of unification. Given terms $t$ and $t'$, a generalization 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 generalization (lgg), that is, a generalization that is more specific than any other generalization. Whereas unification produces most general unifiers that, when applied to two expressions, make them equivalent to the most general common instance of the inputs (Lassez et al., 1988), generalization abstracts the inputs by computing their most specific generalization. As in unification, where the most general unifier (mgu) is of interest, in the sequel we are interested in the least general generalization (lgg) or, as we shall see for the 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 (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, 1980b) 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). Actually, 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. Anti-unification 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 (Dia-

conescu 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 the three have been thoroughly investigated in the literature —see, e.g., (Siekmann, 1989; Baader and Snyder, 1999; Schmidt-Schauss, 1986; Meseguer et al., 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 equational 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 generalization is $f(x,y)$, with $y$ being also a variable. The term $f(f(x,x),y)$ has the advantage of being the most 'specific' or *least general generalization (lgg)* (modulo variable renaming). Moreover, if we have order-sorted information in such a way that constant $a$ is of sort $\mathsf{A}$, constant $b$ is of sort $\mathsf{B}$, but symbol $f$ has two definitions $\mathsf{C} \times \mathsf{C} \to \mathsf{C}$ and $\mathsf{D} \times \mathsf{D} \to \mathsf{D}$ where $\mathsf{A}$ and $\mathsf{B}$ are subsorts of $\mathsf{C}$ and $\mathsf{D}$, then there are four least general generalizations $f(f(x{:}\mathsf{C}, x{:}\mathsf{C}), y{:}\mathsf{C})$, $f(f(x{:}\mathsf{C}, x{:}\mathsf{C}), y{:}\mathsf{D})$ $f(f(x{:}\mathsf{D}, x{:}\mathsf{D}), y{:}\mathsf{C})$, and $f(f(x{:}\mathsf{D}, x{:}\mathsf{D}), y{:}\mathsf{D})$. 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 generalizations $f(x,x,y)$ and $f(a,b,y)$, which are incomparable using associativity and commutativity. Finally, if we combine order-sorted information and equational properties, then there are six least general generalizations $f(a,b,y{:}\mathsf{C})$, $f(a,b,y{:}\mathsf{D})$, $f(x{:}\mathsf{C}, x{:}\mathsf{C}, y{:}\mathsf{C})$, $f(x{:}\mathsf{C}, x{:}\mathsf{C}, y{:}\mathsf{D})$ $f(x{:}\mathsf{D}, x{:}\mathsf{D}, y{:}\mathsf{C})$, and $f(x{:}\mathsf{D}, x{:}\mathsf{D}, y{:}\mathsf{D})$.

The extension of the generalization algorithm to deal with order-sorted functions and equational theories is nontrivial, because of two important reasons. First, as we mentioned the existence and uniqueness of a least general generalization is typically lost. There is a finite and minimal set of *least general generalizations* for two terms, so that any other generalization 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., (Schmidt-Schauss, 1986; Meseguer et al., 1989; 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 generalizations modulo an equational theory $E$ is a difficult task due to the 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. 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 $(\Sigma, E)$ such that any binary function symbol $f \in \Sigma$ can have any combination of the following axioms: (i) *associativity* $(A_f)$ $f(x, f(y, z)) = f(f(x, y), z)$; (ii) *commutativity* $(C_f)$ $f(x, y) = f(y, x)$, and (iii) *identity* $(U_f)$ for a constant symbol, say, $e$, i.e., $f(x, e) = x$ and $f(e, x) = x$. In particular, $f$ may not satisfy any such axioms, which when it happens for all binary symbols $f \in \Sigma$ gives us the standard, 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 —see, e.g., (Martelli and Montanari, 1982; Jouannaud and Kirchner, 1991)— we specify each generalization process by means of an inference system rather than by an imperative-style algorithm.

## Our contribution and plan of the thesis

After some preliminaries, we recall in Chapter 1 a syntactic unsorted generalization algorithm as a special case to motivate later extensions. The main contributions of the thesis can be summarized as follows:

1. An order-sorted generalization algorithm (in Chapter 2). 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 and minimal set of *least general generalizations*, so that any other generalization 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 Chapter 3). 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.

3. An order-sorted modular equational generalization algorithm (in Chapter 4), which combines and refines the inference rules given in Chapters 2 and 3.

4. Formal correctness, completeness, and termination results for all the above generalization algorithms.

5. In Chapter 5, we present the ACUOS system, an implementation of the order-sorted, modular equational generalization algorithm described in Chapter 4. Working with the tool is further illustrated in the example work session described in the Appendix A.

6. Finally, we present some conclusions and directions for future work in Chapter 6.

This paper is an extended and improved version of (Alpuente et al., 2009b,a) 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 Jr., 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-generalizations 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. This is done by adding *features* (or attribute labels) to a sort as argument indicators. 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 but 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). 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" sintaxes that dissolve into a semantically equivalent conjunction of elementary constraints, best defined as a "soup", thanks to the conjuntion 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, 1997) 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 fea-

ture 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). However, 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). This encoding was used to develop a framework, 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 of what we present 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 *preregularity* (or $E$-preregularity 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 in 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 (Meseguer, 1998; Clavel et al., 2007) and the Preliminaries). This means that certain generalization problems are regarded as *incoherent* and have no solution. For example, there is no generalization 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.

# Preliminaries

We follow the classical notation and terminology from (TeReSe, 2003) for term rewriting and from (Meseguer, 1997; Goguen and Meseguer, 1992) for order-sorted equational logic.

We assume an *order-sorted signature* $\Sigma$ with a finite poset of sorts $(\mathsf{S}, \leq)$ and a finite number of function symbols. 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 $\mathsf{s}$, (i.e., if $s$ and $s'$ are sorts in the same connected component, then $[s] = [s']$); and (ii) for each operator declaration $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$ in $\Sigma$, there is also a declaration $f : [\mathsf{s}_1] \times \ldots \times [\mathsf{s}_n] \to [\mathsf{s}]$ in $\Sigma$.

We assume *pre-regularity* of the signature $\Sigma$: for each operator declaration $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$, and for the set $\mathsf{S}_f$ containing sorts $\mathsf{s}'$ appearing in operator declarations of the form $f : \mathsf{s}'_1, \ldots, \mathsf{s}'_n \to \mathsf{s}'$ in $\Sigma$ such that $\mathsf{s}_i \leq \mathsf{s}'_i$ for $1 \leq i \leq n$, then the set $\mathsf{S}_f$ has a least sort. The *unique least sort* of each $\Sigma$-term $t$ is denoted by $LS(t)$. Therefore, 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 $\mathsf{s}$ and $\mathsf{s}'$ in the same connected component, the set of least upper bound sorts of $\mathsf{s}$ and $\mathsf{s}'$, although non necessarily a singleton set, always exists and is denoted by $LUBS(\mathsf{s}, \mathsf{s}')$.

Throughout this paper, we assume that $\Sigma$ 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 $[\mathsf{s}_1] = [\mathsf{s}'_1], \ldots, [\mathsf{s}_n] = [\mathsf{s}'_n], [\mathsf{s}] = [\mathsf{s}']$.

We assume an $\mathsf{S}$-sorted family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s} \in \mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_\mathsf{s}$ 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. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ is the set of terms of sort $\mathsf{s}$, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort $\mathsf{s}$. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding term algebras. For a term $t$, we write $Var(t)$ for the set of all variables in $t$. We assume that $\mathcal{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ for every sort $\mathsf{s}$.

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 $\Lambda$. The subterm of $t$ at position $p$ is $t|_p$ and $t[u]_p$ is the term $t$ where $t|_p$ is replaced by $u$. By $root(t)$ we denote the symbol occurring at the root position of $t$.

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

A $\Sigma$-*equation* is an unoriented pair $t = t'$. An *equational theory* $(\Sigma, E)$ is a set of $\Sigma$-equations. An equational theory $(\Sigma, E)$ is *regular* if for each $t = t' \in E$, we have $Var(t) = Var(t')$. Given $\Sigma$ and a set $E$ of $\Sigma$-equations, 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, 1997).

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 $\sigma$ such that $t\sigma =_E t'$; such a substitution $\sigma$ is said to be an $E$-*matcher* for $t'$ in $t$. The $E$-renaming equivalence $t \simeq_E t'$ (or $\simeq$ if $E$ is empty), holds if there is a renaming $\theta$ such that $t\theta =_E t'$. We write $t <_E t'$ (or $<$ if $E$ is empty) if $t \leq_E t'$ and $t \not\simeq_E t'$.

# 1

# Syntactic Least General Generalization

In order to better present our work, in this chapter we revisit untyped generalization (Plotkin, 1970; Reynolds, 1970; Huet, 1976) 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 Chapter 2 and to the equational setting given in Chapter 3. Throughout this chapter, we assume unsorted terms, i.e., $t \in \mathcal{T}_\Sigma(\mathcal{X})$, with an unsorted signature $\Sigma$. This can be understood as the special case of having only one sort.

Most general unification of a (unifiable) set $M$ of terms is 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" (i.e., $s$ is an instance of $t$, written $t \leq s$, iff there exists $\theta$ such that $t\theta = s$). Formally,

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

and

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

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

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

and

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

The non-deterministic generalization algorithm $\lambda$ of Huet (Huet, 1976) is as follows; also treated in detail in (Lassez et al., 1988). Let $\Phi$ be any bijection between $\mathcal{T}_\Sigma(\mathcal{X}) \times \mathcal{T}_\Sigma(\mathcal{X})$ and a set of variables $V$. The recursive function $\lambda$ 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 $\Phi$ that is used to guarantee that the same disagreements are replaced by the same variable in both terms. Different choices of $\Phi$ may result in different generalizations that are equivalent up to variable renaming.

In the following, we provide a novel set of inference rules for computing the (syntactic) least generalization of two terms, first proposed in (Alpuente et al., 2009b), that uses a local store of already solved generalization sub-problems. The advantage of using such a store is that, differently from the global repository $\Phi$, 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 work. 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.

In our formulation, we represent a generalization problem between terms $s$ and $t$ as a *constraint* $s \overset{x}{\triangleq} t$, where $x$ is a fresh variable that stands for a tentative generalization of $s$ and $t$. By means of this representation, any generalization $w$ of $s$ and $t$ is given by a suitable substitution $\theta$ such that $x\theta = w$.

We compute the least general generalization of $s$ and $t$, written $lgg(s, t)$, by means of a transition system $(Conf, \rightarrow)$ (Plotkin, 2004) where $Conf$ is a set of *configurations* and the transition relation $\rightarrow$ is given by a set of inference rules. Besides the *constraint component*, i.e., a set of constraints of the form $t_i \overset{x_i}{\triangleq} t_{i'}$, and the *substitution component*, i.e., the

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

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

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

Figure 1.1: Rules for least general generalization

partial substitution computed so far, configurations also include the extra constraint component that we call the *store*.

**Definition 1.0.1** *A configuration $\langle CT \mid S \mid \theta\rangle$ consists of three components: (i) the* constraint component CT*, i.e., a conjuntion $s_1 \overset{x_1}{\triangleq} t_1 \wedge \ldots \wedge s_n \overset{x_n}{\triangleq} t_n$ that represents the* set of unsolved constraints*, (ii) the* store component S*, that records the* set of already solved constraints*, and (iii) the* substitution component $\theta$*, that consists of bindings for some variables previously met during the computation.*

Starting from the initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle$, configurations are transformed until a terminal configuration 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, $\theta$ 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 1.1. 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 rule **Decompose** is the syntactic decomposition generating new constraints to be solved.
- The rule **Solve** checks that a constraint $t \overset{x}{\triangleq} t' \in CT$ with $root(t) \neq root(s)$, is not already solved. If not already there, the solved constraint $t \overset{x}{\triangleq} t'$ is added to the store $S$.
- The rule **Recover** checks if a constraint $t \overset{x}{\triangleq} t' \in CT$ 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 1.0.2**

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 generalization of $t$ and $t'$, we apply the inference rules of Figure 1.1. 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 1.2. Note that variable $x_4$ is repeated, to ensure that the least general generalization is obtained.

## 1.1 Termination and Confluence

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

**Theorem 1.1.1 (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 1.1 terminates with a configuration* $\langle \emptyset \mid S \mid \theta \rangle$.

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

Figure 1.2: Computation trace for the (syntactic) generalization of terms $f(g(a),g(y),a)$ and $f(g(b),g(y),b)$

**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 1.1 can be applied, and the application of rules Solve and Recover strictly decreases the size $|CT|$ of the $CT$ component of the lgg configurations at each step, then any derivation necessarily terminates.

■

Note that the inference rules of Figure 1.1 are non–deterministic (i.e., they depend on the chosen constraint of the set $CT$). However, in the following we show that they are confluent up to variable renaming (i.e., the chosen transition is irrelevant for computation of terminal configurations). This justifies the well-known fact that the least general generalization of two terms is unique up to variable renaming (Lassez et al., 1988). In order to prove the confluence up to renaming of the calculus, let us first demonstrate an auxiliary result stating that only (independently) fresh variables $y$ appear in the index positions of the constraints in $CT$ and $S$ components of lgg configurations.

**Lemma 1.1.2 (Uniqueness of Generalization Variables)** *Let $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$ and $x \in \mathcal{X}$. For every derivation $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^*$*

*$\langle CT \mid S \mid \theta \rangle$ stemming from the initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle$ using the inference rules of Figure 1.1, and for every $u \overset{y}{\triangleq} v \in CT$ (similarly $u \overset{y}{\triangleq} v \in S$), the variable $y$ does not appear in any other constraint in $CT$ or $S$, i.e., there are no $u', v' \in \mathcal{T}_\Sigma(\mathcal{X})$ such that $u' \overset{y}{\triangleq} v' \in CT$ or $u' \overset{y}{\triangleq} v' \in S$.*

**Proof.** By induction on the length $n$ of the sequence $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^n \langle CT \mid S \mid \theta \rangle$. If $n = 0$, then the conclusion follows, since $CT = t \overset{x}{\triangleq} t'$ and $S = \emptyset$. If $n > 0$, then we split the derivation into $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^{n-1} \langle CT' \mid S' \mid \theta' \rangle \rightarrow \langle CT \mid S \mid \theta \rangle$ and we consider each inference rule of Figure 1.1 separately:

- **Decompose**. Here $CT' = f(t_1, \ldots, t_n) \overset{x}{\triangleq} f(t'_1, \ldots, t'_n) \wedge CT''$, $S = S'$, $CT = t_1 \overset{x_1}{\triangleq} t'_1 \wedge \ldots \wedge t_n \overset{x_n}{\triangleq} t'_n \wedge CT''$, and $\theta = \theta'\sigma$ where $\sigma = \{x \mapsto f(x_1, \ldots, x_n)\}$, $x_1, \ldots, x_n$ are fresh variables, and $n \geq 0$. By induction hypothesis, $x$ does not appear in $CT''$ and $S'$. Thus, it folllows that $x_1, \ldots, x_n$ do not appear in $CT$ and $S$.

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

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

■

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

**Theorem 1.1.3 (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 1.1 contain a unique solution $\langle \emptyset \mid S \mid \theta \rangle$ up to renaming.*

**Proof.** Given a configuration $\langle t \overset{x}{\triangleq} t' \wedge CT \mid S \mid \theta \rangle$, there is only one possible transition step applicable to $t \overset{x}{\triangleq} t'$ thanks to the non-overlapping inference rules of Figure 1.1. Thus, we must consider the case of having two constraints with the corresponding transitions.

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

- If Decompose is applied to at least one of $t_1 \overset{y}{\triangleq} t_2$ and $t_1' \overset{y'}{\triangleq} t_2'$, then there is no interaction between the constraints, since the Decompose rule is not recording information in the store $S$, and the conclusion follows from the uniqueness of index variables (Lemma 1.1.2). That is, given two inference steps

$$\langle t_1 \overset{y}{\triangleq} t_2 \wedge t_1' \overset{y'}{\triangleq} t_2' \wedge CT \mid S \mid \theta \rangle \rightarrow \langle t_1' \overset{y'}{\triangleq} t_2' \wedge CT_1 \mid S_1 \mid \theta_1 \rangle$$

  and

$$\langle t_1 \overset{y}{\triangleq} t_2 \wedge t_1' \overset{y'}{\triangleq} t_2' \wedge CT \mid S \mid \theta \rangle \rightarrow \langle t_1 \overset{y}{\triangleq} t_2 \wedge CT_2 \mid S_2 \mid \theta_2 \rangle,$$

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

$$\langle t_1' \overset{y'}{\triangleq} t_2' \wedge CT_1 \mid S_1 \mid \theta_1 \rangle \rightarrow \langle CT_{12} \mid S_{12} \mid \theta_{12} \rangle,$$

$$\langle t_1 \overset{y}{\triangleq} t_2 \wedge CT_2 \mid S_2 \mid \theta_2 \rangle \rightarrow \langle CT_{21} \mid S_{21} \mid \theta_{21} \rangle,$$

  and $\langle CT_{12} \mid S_{12} \mid \theta_{12} \rangle \simeq \langle CT_{21} \mid S_{21} \mid \theta_{21} \rangle$. That is, there is a renaming substitution between both configurations thanks to the uniqueness of added index variables.

- If Recover is applied to at least one of $t_1 \overset{y}{\triangleq} t_2$ and $t_1' \overset{y'}{\triangleq} t_2'$, we have the same conclusion, since the Recover rule is not recording information in the store $S$.

- 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

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

  and

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

  we have that

  $$\langle t_1 \overset{y'}{\triangleq} t_2 \wedge CT \mid S \wedge t_1 \overset{y}{\triangleq} t_2 \mid \theta\rangle \rightarrow_{Recover} \langle CT \mid S \wedge t_1 \overset{y}{\triangleq} t_2 \mid \theta\rangle$$

  and

  $$\langle t_1 \overset{y}{\triangleq} t_2 \wedge CT \mid S \wedge t_1 \overset{y'}{\triangleq} t_2 \mid \theta\rangle \rightarrow_{Recover} \langle CT \mid S \wedge t_1 \overset{y'}{\triangleq} t_2 \mid \theta\rangle.$$

  Thus, $\langle CT \mid S \wedge t_1 \overset{y}{\triangleq} t_2 \mid \theta\rangle \simeq \langle CT \mid S \wedge t_1 \overset{y'}{\triangleq} t_2 \mid \theta\rangle$ and the conclusion follows.

■

## 1.2 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, note that for a given constraint $t \overset{x}{\triangleq} t'$, the variable $x$ is a valid generalization of $t$ and $t'$, though generally not the least one.

The first lemma 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.

**Lemma 1.2.1** *Given terms $t$ and $t'$ and a fresh variable $x$ such that $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle CT \mid S \mid \theta\rangle$ using the inference rules of Figure 1.1, then $Index(S \cup CT) \subseteq Ran(\theta)$, and $Ran(\theta) = Var(x\theta)$.*

**Proof.** Immediate by construction.

■

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

**Lemma 1.2.2** *Given terms $t$ and $t'$ and a fresh variable $x$, $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle u \stackrel{y}{\triangleq} v \wedge CT \mid S \mid \theta\rangle$ using the inference rules of Figure 1.1 iff there exists a position $p$ of $t$ and $t'$ such that $t|_p = u$, $t'|_p = v$, and $\forall p' < p$, $root(t|_{p'}) = root(t'|_{p'})$.*

**Proof.** Straightforward by successive application of the inference rule Decompose of Figure 1.1.

■

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

**Definition 1.2.3 (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 states the appropriate connection between the constraints in a derivation and the conflict pairs of the initial configuration.

**Lemma 1.2.4** *Given terms $t$ and $t'$ and a fresh variable $x$, $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle CT \mid u \stackrel{y}{\triangleq} v \wedge S \mid \theta\rangle$ using the inference rules of Figure 1.1 iff there exists a conflict position $p$ of $t$ and $t'$ such that $t|_p = u$ and $t'|_p = v$.*

**Proof.** ($\Rightarrow$) If $u \overset{y}{\triangleq} v \in S$, then there must be two configurations $\langle u \overset{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle$, $\langle CT_2 \mid u \overset{y}{\triangleq} v \wedge S_2 \mid \theta_2\rangle$ such that

$$\begin{array}{rc} & \langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \,\rangle \\ \rightarrow^* & \langle u \overset{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle \\ \rightarrow & \langle CT_2 \mid u \overset{y}{\triangleq} v \wedge S_2 \mid \theta_2\rangle \\ \rightarrow^* & \langle \emptyset \mid S \mid \theta\rangle, \end{array}$$

$u \overset{y}{\triangleq} v \notin S_1$, $u \overset{y}{\triangleq} v \notin CT_2$, and $root(u) \neq root(v)$. By Lemma 1.2.2, 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 1.2.2, there is a configuration $\langle u \overset{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle$ such that $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \,\rangle \rightarrow^* \langle u \overset{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle$, $u \overset{y}{\triangleq} v \notin S_1$, and $root(u) \neq root(v)$. Then, the inference rule Solve is applied, i.e., $\langle u \overset{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle \rightarrow \langle CT_1 \mid u \overset{y}{\triangleq} v \wedge S_1 \mid \theta_1\rangle$ and the constraint $u \overset{y}{\triangleq} v$ 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 generalization term.

**Lemma 1.2.5** *Given terms $t$ and $t'$ and a fresh variable $x$, $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figure 1.1 iff $x\theta$ is a generalization of $t$ and $t'$.*

**Proof.** By structural induction on the term $x\theta$. If $x\theta = x$, then $\theta = id$ and the conclusion follows. If $x\theta = f(u_1, \ldots, u_k)$, 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 generalization of $t_i$ and $t'_i$, for each $i$. Now, if there is no variable shared between two different $u_i$, 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 1.2.4, there are conflict positions $p_i$ in $t_i$ and $t'_i$, and $p_j$

in $t_j$ and $t'_j$ such that $t_i|_{p_i} = t_j|_{p_j}$ and $t'_i|_{p_i} = t'_j|_{p_j}$. Thus, the conclusion follows.

■

Finally, correctness and completeness are proved as follows.

**Theorem 1.2.6 (Correctness and Completeness)** *Given terms $t$ and $t'$ and a fresh variable $x$, $u$ is the lgg of $t$ and $t'$ iff $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ using the inference rules of Figure 1.1 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 1.2.5, $x\theta$ is a generalization 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 $\rho$ which is not a variable renaming such that $x\theta\rho = u$. By Lemma 1.2.1, $Ran(\theta) = Var(x\theta)$, hence we can choose $\rho$ with $Dom(\rho) = Var(x\theta)$. Now, since $\rho$ 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 1.2.4 and 1.2.1. 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 1.2.4 and 1.2.1, the claim is proved.

■

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.

# 2

# Order–sorted Least General Generalizations

In this chapter, we generalize the unsorted generalization algorithm presented in Chapter 1 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 $\langle t \stackrel{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle$ where $[\mathsf{s}] = [LS(t)] = [LS(t')]$, configurations are transformed until a terminal configuration $\langle \emptyset \mid S \mid \theta \rangle$ is reached. 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 $\delta$ is called *downgrading* if each binding is of the form $x{:}\mathsf{s} \mapsto x'{:}\mathsf{s}'$, where $x$ and $x'$ are variables and $\mathsf{s}' \leq \mathsf{s}$.

The transition relation $\rightarrow$ is given by the smallest relation satisfying the rules in Figure 2.1. The meaning of these rules is as follows.

- The rule **Decompose** 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.

- The rule **Recover** reuses a previously solved constraint, similarly to to the corresponding unsorted rule of Figure 1.1.

- The rule **Solve** checks that a constraint $t \stackrel{y}{\triangleq} t' \in C$, with $root(s) \neq root(t)$, is not already solved. Then the solved constraint $t \stackrel{y}{\triangleq} t'$ is added to the store $S$, and the substitution $\{x \mapsto z\}$ is composed

**Decompose**

$$\frac{f \in (\Sigma \cup \mathcal{X}) \wedge f : [\mathsf{s_1}] \times \ldots \times [\mathsf{s_n}] \to [\mathsf{s}]}{\langle f(t_1, \ldots, t_n) \overset{x:[\mathsf{s}]}{\triangleq} f(s_1, \ldots, s_n) \wedge C \mid S \mid \theta \rangle \to \langle t_1 \overset{x_1:[\mathsf{s_1}]}{\triangleq} s_1 \wedge \ldots \wedge t_n \overset{x_n:[\mathsf{s_n}]}{\triangleq} s_n \wedge C \mid S \mid \theta\sigma \rangle}$$

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**

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

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

**Recover**

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

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

Figure 2.1: Rules for order–sorted least general generalization.

with the substitution part, where $z$ is a fresh variable with sort in the $LUBS$ of the least sorts of both terms. Note that this is the only additional source of non-determinism (besides the choice of the constraint to work on) in our inference rules, in contrast to Figure 1.1. 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) generalizations $x\theta_1, \ldots, x\theta_n$.

**Example 2.0.7**

Let $t = f(x{:}\mathsf{A})$ and $t' = f(y{:}\mathsf{B})$ 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 2.3. The typed definition of $f$ is $f : \mathsf{E} \to \mathsf{E}$. Starting from the initial configuration $\langle f(x{:}\mathsf{A}) \overset{z:\mathsf{E}}{\triangleq} f(y{:}\mathsf{B}) \mid \emptyset \mid id \rangle$, we apply the inference rules of Figure 2.1 and the substitutions obtained by the lgg algorithm are $\theta_1 = \{z{:}\mathsf{E} \mapsto f(z_2{:}\mathsf{C}), z_1{:}\mathsf{E} \mapsto z_2{:}\mathsf{C}\}$ and $\theta_2 = \{z{:}\mathsf{E} \mapsto$

$$
\begin{array}{c}
lgg(f(x{:}\mathsf{A}), f(y{:}\mathsf{B})) \\
\downarrow \text{ Initial Configuration} \\
\langle f(x{:}\mathsf{A}) \overset{z:\mathsf{E}}{\triangleq} f(y{:}\mathsf{B}) \mid \emptyset \mid id \rangle \\
\downarrow \text{ Decompose} \\
\langle x{:}\mathsf{A} \overset{z_1:\mathsf{E}}{\triangleq} y{:}\mathsf{B} \mid \emptyset \mid \{z{:}\mathsf{E} \mapsto f(z_1{:}\mathsf{E})\} \rangle \\
\swarrow \text{ Solve } \searrow \\
\langle \emptyset \mid x{:}\mathsf{A} \overset{z_2:\mathsf{C}}{\triangleq} y{:}\mathsf{B} \mid \{z{:}\mathsf{E} \mapsto f(z_2{:}\mathsf{C}), z_1{:}\mathsf{E} \mapsto z_2{:}\mathsf{C}\} \rangle \quad \langle \emptyset \mid x{:}\mathsf{A} \overset{z_3:\mathsf{D}}{\triangleq} y{:}\mathsf{B} \mid \{z{:}\mathsf{E} \mapsto f(z_3{:}\mathsf{D}), z_1{:}\mathsf{E} \mapsto z_3{:}\mathsf{D}\} \rangle
\end{array}
$$

Figure 2.2: Computation trace for the order–sorted generalization of terms $f(x)$ and $f(y)$

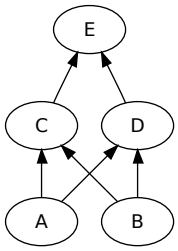


Figure 2.3: Sort hierarchy

$f(z_3{:}\mathsf{D}), z_1{:}\mathsf{E} \mapsto z_3{:}\mathsf{D}\}$. Note that $\theta_1$ and $\theta_2$ are incomparable, so that we have two posible lggs where $(z{:}\mathsf{E})\theta_1 = f(z_2{:}\mathsf{C})$ and $(z{:}\mathsf{E})\theta_2 = f(z_3{:}\mathsf{D})$. The computation of both solutions is illustrated in Figure 2.2.

## 2.1 Termination and Confluence

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

**Theorem 2.1.1 (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 2.1 where* $\mathsf{s} = [LS(t)] = [LS(t')]$ *terminates with a configuration* $\langle \emptyset \mid S \mid \theta \rangle$.

**Proof.** Similar to the proof of Theorem 1.1.1.

■

The transition system $(Conf, \rightarrow)$ is no longer confluent, as shown in Example 2.0.7. However, confluence can be recovered under appropriate conditions.

**Definition 2.1.2 (Top-sorted Constant)** *Given an order-sorted signature $\Sigma$, we say that a constant $c : nil \rightarrow \mathsf{s}$ is* top-sorted *if $\mathsf{s} = [\mathsf{s}]$.*

**Definition 2.1.3 (Top-sorted Variable)** *A variable $x{:}\mathsf{s}$ is called* top-sorted *if $\mathsf{s} = [\mathsf{s}]$.*

**Definition 2.1.4 (Top-sorted Term)** *A term $t$ is called* top-sorted *if every variable and every constant in $t$ are top-sorted.*

The following result uses the assumption of a kind-completed order-sorted signature described in the Preliminaries.

**Lemma 2.1.5** *Given a top-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 2.1.6** *Given two top-sorted terms $t, t'$, $LUBS(LS(t), LS(t')) = LS(t) = LS(t') = [LS(t)] = [LS(t')]$.*

**Proof.** By Lemma 2.1.5, $LS(t) = [LS(t)]$ 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 2.1.7 (Confluence)** *The set of derivations stemming from an initial configuration $\langle t \overset{x:\mathsf{s}}{\triangleq} t' \mid \emptyset \mid id \rangle$ using the inference rules of Figure 2.1 where $t$ and $t'$ are top-sorted terms and $\mathsf{s} = [LS(t)] = [LS(t')]$, contain a unique solution $\langle \emptyset \mid S \mid \theta \rangle$ up to renaming.*

**Proof.** Similar to the proof of Theorem 1.1.3, but taking into account that Lemma 2.1.6 ensures that there is no non-determinism involved in the application of the inference rule Solve.

■

## 2.2 Order-sorted lgg computation by subsort specialization

Even if the set of least general generalizations of two terms is not generally a singleton, there is still a unique top-sorted generalization that can just be specialized into the appropriate subsorts. This enables a different approach to computing order-sorted least general generalizations 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. Obviously, the set of inference rules of Figure 2.1 has a better performance than this alternative method of first upgrading variables, computing the standard lggs, and then downgrading variables, since the inference rules of Figure 2.1 detect sort-based failures much earlier. Indeed, we do not use this approach in practice, but we only use it for the proofs of correctness and completeness of the inference rules given in Section 2.3 below. Note that this proof schema for correctness and completeness of inference rules is useful here for the order-sorted generalization and also for the order-sorted $E$-generalization of Chapter 4 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 1.2.3 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 2.2.1 (Top-sorted Generalization)** *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 $p^i_1, \ldots, p^i_{n_i}$ 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*

$$tsg(t, t') = ((t[x_1{:}\mathsf{s}_1]_{p^1_1,\ldots,p^1_{n_1}})\cdots)[x_k{:}\mathsf{s_k}]_{p^k_1,\ldots,p^k_{n_k}}$$

*where $x_1{:}\mathsf{s}_1, \ldots, x_k{:}\mathsf{s_k}$ are fresh variables.*

**Example 2.2.2**
Let us consider the terms $t = f(x{:}\mathsf{A})$ and $t' = f(y{:}\mathsf{B})$ of Example 2.0.7. We have that $tsg(t,t') = f(z{:}\mathsf{E})$, since there is only one conflict pair $(x{:}\mathsf{A}, y{:}\mathsf{B})$ 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 2.2.3 (Sort-specialized Generalization)** *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 $t'$. We define*

$$\begin{aligned} \textit{sort-down-subs}(t,t') = \{\rho \mid\ & Dom(\rho) = \{x_1{:}\mathsf{s_1}, \ldots, x_k{:}\mathsf{s_k}\} \\ & \wedge\ \forall 1 \leq i \leq k, \rho(x_i{:}\mathsf{s_i}) = x_i{:}\mathsf{s'_i} \\ & \wedge\ \mathsf{s'_i} \in LUBS(LS(u_i), LS(v_i))\} \end{aligned}$$

*where all the $x_i{:}\mathsf{s'_i}$ are fresh variables. The set of sort-specialized generalizations of $t$ and $t'$ is defined as $ssg(t,t') = \{tsg(t,t')\rho \mid \rho \in \textit{sort-down-subs}(t,t')\}$.*

**Example 2.2.4**
Continuing Example 2.2.2, we have that $ssg(t,t') = \{f(z{:}\mathsf{C}), f(z{:}\mathsf{D})\}$, with $\textit{sort-down-subs}(t,t') = \{\{x{:}\mathsf{E} \mapsto z{:}\mathsf{C}\}, \{x{:}\mathsf{E} \mapsto z{:}\mathsf{D}\}\}$.

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

**Theorem 2.2.5** *Given terms $t$ and $t'$ such that $[LS(t)] = [LS(t')]$, it holds that 1) $tsg(t,t')$ is a generalization 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 generalization of $t$ and $t'$, since for each conflict pair $(s,s')$, the term $tsg(t,t')$ contains a variable at the corresponding conflict position of $t$ and $t'$ which has the top sort associated to $s$ and $s'$.

We prove that $sgg(t,t')$ provides a minimal complete set of order-sorted lggs by contradiction. First, let us prove that it is complete by assuming that there is a generalization $u$ of $t$ and $t'$ s.t. there is no

$u' \in ssg(t,t')$ with $u \leq u'$. By definition of $tsg(t,t')$, we either have that $u \leq tsg(t,t')$ or $tsg(t,t') \leq u$. If $u \leq tsg(t,t')$, there must be a term $u' \in ssg(t,t')$ such that $u \leq u'$. If $tsg(t,t') \leq u$, then at least one of the variables $x_i{:}\mathsf{s_i}$ of a conflict pair must have been instantiated with a variable $z{:}\mathsf{s}$ such that $\mathsf{s} \leq \mathsf{s_i}$, but then 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 generalizations $u, u'$ of $t$ and $t'$ s.t. $u \in ssg(t,t')$, $u' \in ssg(t,t')$, and $u < u'$. If $u < u'$, then at least one of the variables $x_i{:}\mathsf{s_i}$ of $u$ corresponding to a conflict pair $(u_i, v_i)$ must have been instantiated with a variable $x'_i{:}\mathsf{s'_i}$ of $u'$ such that $\mathsf{s'_i} < \mathsf{s_i}$, which is impossible by definition of $LUBS(LS(u_i), LS(v_i))$.

■

## 2.3 Correctness and Completeness of the order-sorted lgg calculus

Before proving correctness and completeness of the order-sorted lgg calculus given in Figure 2.1, we provide some auxiliary notions and lemmata.

The first lemma links the constraints with positions in terms $t$ and $t'$ of a generalization problem.

**Lemma 2.3.1** *Given terms $t$ and $t'$ such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, and a fresh variable $x{:}[\mathsf{s}]$, $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle u \overset{y:[\mathsf{s'}]}{\triangleq} v \wedge CT \mid S \mid \theta\rangle$ using the inference rules of Figure 2.1 iff 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 application of the Decompose inference rule of Figure 2.1.

■

The following lemma links constraints already solved (and thus saved in the store) with conflict positions of terms $t$ and $t'$ of a generalization problem.

**Lemma 2.3.2** *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 2.1, the constraint $u \overset{y:\mathsf{s}'}{\triangleq} v$ belongs to $S$ iff 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 $\mathsf{s}''$ and two configurations $\langle u \overset{y:\mathsf{s}''}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle$, $\langle CT_2 \mid u \overset{y:\mathsf{s}'}{\triangleq} v \wedge S_2 \mid \theta_2\rangle$ such that

$$\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle u \overset{y:\mathsf{s}''}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle \rightarrow \langle CT_2 \mid u \overset{y:\mathsf{s}'}{\triangleq} v \wedge S_2 \mid \theta_2\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle,$$

$u \overset{y:\mathsf{s}''}{\triangleq} v \notin S_1$, $u \overset{y:\mathsf{s}'}{\triangleq} v \notin CT_2$, $\mathsf{s}' \leq \mathsf{s}''$, and $root(u) \neq root(v)$. By Lemma 2.3.1, 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 2.3.1, there exist a sort $[\mathsf{s}'']$ and a configuration $\langle u \overset{y:[\mathsf{s}'']}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle$ such that $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle u \overset{y:[\mathsf{s}'']}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle$, $u \overset{y:\mathsf{s}''}{\triangleq} v \notin S_1$, and $root(u) \neq root(v)$. Then, the inference rule Solve is applied, i.e., $\langle u \overset{y:[\mathsf{s}'']}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle \rightarrow \langle CT_1 \mid u \overset{y:\mathsf{s}'}{\triangleq} v \wedge S_1 \mid \theta_1\rangle$, and $\mathsf{s}' \in LUBS(LS(u), LS(v))$. Thus, 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 2.3.3** *Given terms $t$ and $t'$ such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, for all $S$ and $\theta$ 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 2.1, there exists a downgrading substitution $\delta$ such that $tsg(t, t')\delta = (x{:}[\mathsf{s}])\theta$.*

**Proof.** By successive application of the Decompose inference rule of Figure 2.1.

■

**Theorem 2.3.4 (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 $t'$ iff there exists $S$ and $\theta$ 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 2.1 and $u \simeq (x{:}[\mathsf{s}])\theta$.*

**Proof.** We reason by contradiction.

($\Rightarrow$) Let us consider a store $S$ and substitution $\theta$ such that there is no term $u$ and renaming $\rho$ with $u\rho = (x{:}[\mathsf{s}])\theta$. By Theorem 2.2.5, $tsg(t,t') \leq u$ with a downgrading substitution $\delta_u$, i.e., $tsg(t,t')\delta_u = u$. By Lemma 2.3.3, $tsg(t,t') \leq (x{:}[\mathsf{s}])\theta$ with a downgrading substitution $\delta$, 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 $\delta$ and one binding $x{:}[\mathsf{s}] \mapsto x''{:}\mathsf{s}''$ in $\delta_u$ s.t. either $\mathsf{s}' < \mathsf{s}''$, $\mathsf{s}'' < \mathsf{s}'$, or $[\mathsf{s}'] \neq [\mathsf{s}'']$. 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 2.3.2, and $[\mathsf{s}'] \neq [\mathsf{s}'']$ contradicts both that $u$ is a lgg of $t$ and $t'$ and Lemma 2.3.2.

($\Leftarrow$) This case can be proven similarly.

■

# 3

# 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 $E$-generalizations (Baader, 1991).

For a set $M$ of terms, we define the set of most specific generalizations of $M$ modulo $E$ as the set of *maximal lower bounds* of $M$ under $<_E$, i.e., $lgg_E(M) = \{u \mid \forall m \in M, u \leq_E m \wedge \nexists u' \ (u <_E u' \wedge \forall m \in M, u' \leq_E m)\}$.

**Example 3.0.5**

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

## 3.1 Recursively enumerating the least general generalizations 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 generalizations of $t$ and $s$. For this, we only need to recursively enumerate all pairs of terms $(u, u')$ with $t =_E u$ and $s =_E u'$ and compute $lgg(u, u')$. Of course, this set $gen_E(t, s)$ may easily be infinite. However, if the

theory $E$ has the additional property that each $E$-equivalence class is *finite* and can be effectively generated, then the above process becomes a terminating *algorithm*, generating a finite complete set of generalizations of $t$ and $s$.

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

**Definition 3.1.1** *Let $t$ and $s$ be terms and let $E$ be an equational theory. A complete set of generalizations of $t$ and $s$ modulo $E$, denoted by $gen_E(t, t')$, is defined as follows:*

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

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

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

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

Our modular $E$-generalization algorithm defined below computes a complete set of generalizations, i.e., the set $gen_E(t, t')$, that must be filtered out to obtain the least general generalizations, i.e., the set $lgg_E(t, t')$. Let us prove that the set $\text{gen}_E(t, t')$ is a complete set of $E$-lggs.

**Lemma 3.1.2** *Given terms $t$ and $t'$ in an equational theory $E$, if $u$ is an lgg 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 $s$ modulo $E$ and assume that there is no $u' \in gen_E(t, t')$ such that $u' \simeq_E u$. Since $u \leq_E t$ and $u \leq_E s$, there exist substitutions $\sigma_t$ and $\sigma_s$ such that $u\sigma_t =_E t$ and $u\sigma_s =_E s$. But then, $u \in lgg(u\sigma_t, u\sigma_s)$ (i.e., without making use of the

equations $E$) and, by definition, $u \in gen_E(t, s)$, which contradicts the assumption.

■

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

**Theorem 3.1.3** *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 $t'$ (up to renaming).*

**Proof.** Lemma 3.1.2 ensures that $gen_E(t, t')$ is a complete set of lggs. Minimality of the set $lgg_E(t, t')$ is ensured by maximality of the relation $<_E$.

■

However, note that in general the relation $t <_E t'$ is *undecidable*, so that the above set, although definable at the mathematical level, might not be effectively computed. Nevertheless, when: (i) each $E$-equivalence class is *finite* and can be effectively generated, and (ii) there is an $E$-matching algorithm, then we also have an effective algorithm for computing $lgg_E(t, s)$, since the relation $t \leq_E t'$ 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, although horribly inefficient, procedure to compute a finite, minimal, and complete set of least general generalizations $lgg_E(t, s)$, 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, minimal, terminating, sound, and complete algorithm for equational theories containing different axioms such as associativity, commutativity, and identity (and

their combinations). This algorithm computes the set $gen_E(t, t')$ modulo $E$ and renaming. The set $lgg_E(t, s)$ of least general $E$-generalizations can be computed as in Definition 3.1.1. That is: first a complete set of $E$-generalizations 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 $<_E$. We consider that a given function symbol $f$ in the signature $\Sigma$ 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 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 3.2, then, $ax(f) = \{C_f\}$ in Section 3.3, then, $ax(f) = \{A_f\}$ in Section 3.4, then, $ax(f) = \{A_f, C_f\}$ in Section 3.5, and finally, $U_f \in ax(f)$ in Section 3.6. In each section, proofs of correctness and completeness are very similar to the ones in Section 1.2 and, thus, we define a key notion of conflict pair for each equational property (i.e., commutative conflict pairs, associative conflict pairs, associative-commutative conflict pairs, and identity conflict pairs) which is 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.

## 3.2 Basic inference rules for least general $E$–generalization

Let us start with a set of basic rules that are the equational version of the syntactic generalization rules of Chapter 1. The rule $Decompose_E$ applies to function symbols obeying no axioms, $ax(f) = \emptyset$. Specific rules for decomposing constraints involving terms that are rooted by symbols obeying equational axioms, such as ACU and their combinations, are given below.

Concerning the rules $Solve_E$ and $Recover_E$, the main difference w.r.t. the corresponding syntactic generalization rules given in Chapter 1 is in

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

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

**Recover**$_E$
$$\frac{root(t) \neq root(t') \wedge \exists y : t \overset{y}{\triangleq} t' \in^E S}{\langle t \overset{x}{\triangleq} t' \wedge CT \mid S \mid \theta\rangle \rightarrow \langle CT \mid S \mid \theta\sigma\rangle}$$
where $\sigma = \{x \mapsto y\}$

Figure 3.1: Basic inference rules for least general $E$–generalization

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

Finally, regarding the rule $Solve_E$, note that this rule cannot be applied to any constraint $t \overset{x}{\triangleq} s$ such that either $t$ or $s$ are rooted by a function symbol $f$ with $U_f \in ax(f)$. For function symbols with an identity element, a specially–tailored rule $Expand_U$ is given in Section 3.6 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 generalization $f(y, z)$ 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 Chapter 1.

**Theorem 3.2.1 (Termination)** *Given an equational theory $(\Sigma, E)$, $\Sigma$-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 3.1 terminates with a configuration*

$\langle \emptyset \mid S \mid \theta \rangle$.

**Proof.** It follows directly from Theorem 1.1.1. ■

**Theorem 3.2.2 (Correctness and Completeness)** *Given an equational theory $(\Sigma, E)$, $\Sigma$-terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free, and a fresh variable $x$, then $u \in gen_E(t, t')$ iff there is $u'$ in $\{x\theta \mid \langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle \emptyset \mid S \mid \theta \rangle\}$ using the inference rules of Figure 3.1 such that $u \simeq u'$.*

**Proof.** It follows directly from Theorem 1.2.6.

■

Note that the basic inference rules of Figure 3.1 are confluent when $E = \emptyset$, according to Theorem 1.1.3, but the inference system of Chapter 4 combining 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) generalizations $x\theta_1, \ldots, x\theta_n$.

## 3.3 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 3.2, for dealing with commutativity function symbols. This inference rule replaces the syntactic decomposition inference rule for the case of a binary commutative symbol $f$, i.e., the 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 3.3.1**

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

**Decompose**$_C$

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

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

Figure 3.2: Decomposition rule for a commutative function symbol $f$

Termination is straightforward.

**Theorem 3.3.2 (Termination)** *Given an equational theory $(\Sigma, E)$, $\Sigma$-terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or commutative, 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 3.1 and 3.2 terminates with a configuration $\langle \emptyset \mid S \mid \theta\rangle$.*

**Proof.** Similar to the proof of Theorem 1.1.1 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 1.2.3 we introduce the auxiliary concept of *commutative conflict pair*, and prove some useful results for this case.

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

**Lemma 3.3.3 (Uniqueness of Generalization Variables)** *Lemma 1.1.2 holds for $t \stackrel{x}{\triangleq} t'$ when the symbols in $t$, $t'$ are free or commutative, for the inference rules of Figures 3.1 and 3.2.*

The first lemma 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.

**Lemma 3.3.4** *Given terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or commutative, and a fresh variable $x$ such that $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle CT \mid S \mid \theta\rangle$ using the inference rules of Figures 3.1 and 3.2, then $Index(S \cup CT) \subseteq Ran(\theta)$, and $Ran(\theta) = Var(x\theta)$.*

**Proof.** Immediate by construction.

■

The following lemma 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 $depth(\Lambda) = 0$ 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., $p|_0 = \Lambda$, $(i.p)|_k = i.(p|_{k-1})$ if $k > 0$. For instance, for $p = 1.2.1.3$, $p|_3 = 1.2.1$.

**Lemma 3.3.5** *Given terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or commutative, and a fresh variable $x$, $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle u \stackrel{y}{\triangleq} v \wedge CT \mid S \mid \theta\rangle$ using the inference rules of Figures 3.1 and 3.2 iff there exist a position $p \in Pos(t)$ and a position $p' \in Pos(t')$ such that $t|_p = u$, $t'|_{p'} = v$, $depth(p) = depth(p')$, and $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$.*

**Proof.** Straightforward by successive application of the inference rule Decompose of Figure 1.1 and the inference rule $\text{Decompose}_C$ of Figure 3.2.

■

**Definition 3.3.6 (Commutative Conflict Pair)** *Given terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or commutative, the pair $(u, v)$ is called a* commutative conflict pair of $t$ and $t'$ *iff $u \neq_E v$ and there exist at least one position $p \in Pos(t)$ and at least one position $p' \in Pos(t')$ such that $t|_p = u$, $t'|_{p'} = v$, $depth(p) = depth(p')$, and $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$.*

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

**Lemma 3.3.7** *Given terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or commutative, and a fresh variable $x$, $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle CT \mid u \stackrel{y}{\triangleq} v \wedge S \mid \theta\rangle$ using the inference rules of Figures 3.1 and 3.2 iff $(u, v)$ is a commutative conflict pair of $t$ and $t'$.*

**Proof.** ($\Rightarrow$) If $u \overset{y}{\triangleq} v \in S$, then there must be two configurations $\langle u \overset{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle$, $\langle CT_2 \mid u \overset{y}{\triangleq} v \wedge S_2 \mid \theta_2\rangle$ such that

$$\begin{array}{rc} & \langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \,\rangle \\ \rightarrow^* & \langle u \overset{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle \\ \rightarrow & \langle CT_2 \mid u \overset{y}{\triangleq} v \wedge S_2 \mid \theta_2\rangle \\ \rightarrow^* & \langle \emptyset \mid S \mid \theta\rangle, \end{array}$$

$u \overset{y}{\triangleq} v \notin S_1$, $u \overset{y}{\triangleq} v \notin CT_2$, and $root(u) \neq root(v)$. By Lemma 3.3.5, there exists a position $p \in Pos(t)$ and a position $p' \in Pos(t')$ such that $t|_p = u$, $t'|_{p'} = v$, $depth(p) = depth(p')$, and $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$. Therefore, $(u, v)$ is a commutative conflict pair.

($\Leftarrow$) By Lemma 3.3.5, there is a configuration $\langle u \overset{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle$ such that $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \,\rangle \rightarrow^* \langle u \overset{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle$, $u \overset{y}{\triangleq} v \notin S_1$, and $root(u) \neq root(v)$. Then, the inference rule Solve is applied, i.e., $\langle u \overset{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1\rangle \rightarrow \langle CT_1 \mid u \overset{y}{\triangleq} v \wedge S_1 \mid \theta_1\rangle$ and $u \overset{y}{\triangleq} v$ 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 generalization term.

**Lemma 3.3.8** *Given terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or commutative, and a fresh variable $x$, $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$, using the inference rules of Figures 3.1 and 3.2 iff $x\theta$ is a generalization of $t$ and $t'$ modulo commutativity.*

**Proof.** By structural induction on the term $x\theta$. If $x\theta = x$, then $\theta = id$ and the conclusion follows. If $x\theta = f(u_1, \ldots, u_k)$ and $f$ is free, then the inference rule Decompose$_E$ of Figure 3.1 is applied and we have that $t = f(t_1, \ldots, t_k)$ and $t' = f(t'_1, \ldots, t'_k)$. If $x\theta = f(u_1, \ldots, u_k)$ and $f$ is commutative, then the inference rule Decompose$_C$ of Figure 3.2 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 $t' = f(t'_2, t'_1)$. By induction hypothesis, $u_i$ is

a generalization of $t_i$ and $t'_i$, for each $i$. 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 3.3.7, there is a 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 3.3.9 (Correctness and Completeness)** *Given an equational theory $(\Sigma, E)$, $\Sigma$-terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or commutative, and $x$ a fresh variable, then $u \in gen_E(t, t')$ iff there is $u'$ in $\{x\theta \mid \langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle\}$ using the inference rules of Figures 3.1 and 3.2 such that $u \simeq_E u'$.*

**Proof.** By contradiction. By Lemma 3.3.8, $x\theta$ is a generalization of $t$ and $t'$. If $x\theta$ is not a generalization of $t$ and $t'$ up to renaming, then there is a term $u$ which is a generalization of $t$ and $t'$ and a substitution $\rho$ which is not a variable renaming such that $x\theta\rho =_E u$. By Lemma 3.3.4, $Ran(\theta) = Var(x\theta)$, hence we can choose $\rho$ with $Dom(\rho) = Var(x\theta)$. Since $\rho$ 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 positions $p, p'$ in $u$ such that $u|_p = z = u|_{p'}$. Moreover, there is a position $q$ in $x\theta$ such that $(x\theta)|_q = y$ and the pair $(y, z)$ is a conflict pair of $x\theta$ and $u$. Similarly there is a position $q'$ in $x\theta$ such that $(x\theta)|_{q'} = y'$ and the pair $(y', z)$ is a conflict pair of $x\theta$ and $u$. But this also means that there is a position $q_t$ in $t$ such that $t|_{q_t} = w_1$ and the pair $(w_1, z)$ is a conflict pair of $t$ and $u$; and there is a position $q'_{t'}$ in $t'$ such that $t|_{q'_{t'}} = w_2$ and the pair $(w_2, z)$ is a conflict pair of $t'$ and $u$. But this is impossible by Lemmas 3.3.7 and 3.3.4. In case (2), there is a position $p$ such that $(x\theta)|_p = y$ and, since $y\rho = v$ and $v$ is a

non-variable term, then $p$ is not involved in any conflict pair of $t$ and $t'$. But this is again impossible by Lemmas 3.3.7 and 3.3.4.

■

We recall again that in general the inference rules of Figures 3.1 and 3.2 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 generalizations $x\theta_1, \ldots, x\theta_n$.

## 3.4 Least general generalization modulo $A$

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

The $Decompose_A$ rule is given in Figure 3.3. 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

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

Given an associative symbol $f$ and a term $f(t_1, \ldots, t_n)$ we call *$f$-alien terms* (or simply *alien terms*) to those terms among $t_1, \ldots, t_n$ that are not rooted by $f$. In the following, being $f$ an associative poly-varyadic symbol, by convention $f(t)$ represents the term $t$ itself, since symbol $f$ needs at least two arguments. The inference rule of Figure 3.3 replaces the syntactic decomposition inference rule for the case of an associative function symbol $f$, where all *prefixes* of $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_m$ are considered. Note that this rule is (don't know) non-deterministic, hence all possibilities must be explored.

This inference rule for associativity is better than generating all terms in the corresponding equivalence class, as explained in Chapter 3, 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 $f(t_1, \ldots, t_n)$.

We give the rule $Decompose_A$ for the case when, in the generalization problem $f(t_1, ..., t_n) \stackrel{x}{\triangleq} f(s_1, ..., s_m)$, we have that $n \geq m$. For the other

**Decompose$_A$**

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

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

Figure 3.3: Decomposition rule for an associative (non–commutative) function symbol $f$

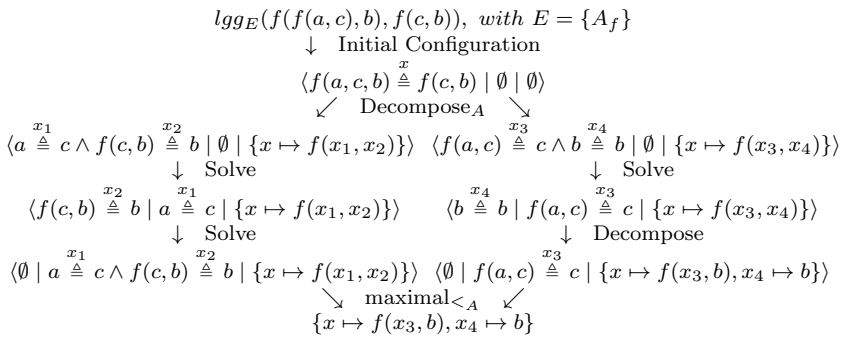


Figure 3.4: Computation trace for the A–generalization of terms $f(f(a,c),b)$ and $f(c,b))$.

way around, i.e., $n < m$, a similar rule would be needed, that we omit since it is entirely similar. The following example illustrates the least general generalization modulo A.

**Example 3.4.1**

Let $t = f(f(a,c),b)$ and $t' = f(c,b)$ be two terms where $f$ is associative, i.e., $ax(f) = \{A_f\}$. By applying the rules $Solve_E$, $Recover_E$, and $Decompose_A$ above, we end in a terminal configuration $\langle \emptyset \mid S \mid \theta\rangle$, where $\theta = \{x \mapsto f(x_3,b), x_4 \mapsto b\}$, thus we obtain that the lgg modulo $A$ of $t$ and $t'$ is $f(x_3,b)$. The computation trace is shown in Figure 3.4.

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 3.4.2**

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., $ax(f) = \{A_f\}$. 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 are more general terms.

Termination is straightforward.

**Theorem 3.4.3 (Termination)** *Given an equational theory $(\Sigma, E)$, $\Sigma$-terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative, and $x$ a fresh variable, every derivation stemming from an initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle$ using the inference rules of Figures 3.1 and 3.3 terminates with a configuration $\langle \emptyset \mid S \mid \theta \rangle$.*

**Proof.** Similar to the proof of Theorem 1.1.1 by simply considering the flattened versions of the terms. ■

In order to prove correctness and completeness of the lgg calculus modulo A, similarly to Definitions 1.2.3 and 3.3.6, we introduce the auxiliary concept of 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 $CT$ and $S$ components of lgg configurations.

**Lemma 3.4.4 (Uniqueness of Generalization Variables)** *Lemma 1.1.2 holds*

*for $t \overset{x}{\triangleq} t'$ when the symbols in $t$, $t'$ are free or associative, for the inference rules of Figures 3.1 and 3.3.*

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.

**Lemma 3.4.5** *Given terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative, and a fresh variable $x$ such that $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle CT \mid S \mid \theta \rangle$ using the inference rules of Figures 3.1 and 3.3, then $Index(S \cup CT) \subseteq Ran(\theta)$, and $Ran(\theta) = Var(x\theta)$.*

**Proof.** Immediate by construction.

■

The following lemma 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.

**Lemma 3.4.6** *Given flattened terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative, and a fresh variable $x$, $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle u \overset{y}{\triangleq} v \wedge CT \mid S \mid \theta\rangle$ using the inference rules of Figures 3.1 and 3.3 iff there exists a position $p \in Pos(t)$ and a position $p' \in Pos(t')$ such that either:*

1. *$t|_p = u$, $t'|_{p'} = v$, $depth(p) = depth(p')$, and $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$;*
2. *$t|_p = u$, $v = f(v_1, \ldots, v_n)$, $t'|_{p'} = f(w_1, \ldots, w_m, v_1, \ldots, v_n, w'_1, \ldots, w'_{m'})$, $f$ is associative, $depth(p) = depth(p') + 1$, and $\forall 1 \leq i \leq depth(p')$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$;*
3. *$u = f(u_1, \ldots, u_n)$, $t|_p = f(w_1, \ldots, w_m, u_1, \ldots, u_n, w'_1, \ldots, w'_{m'})$, $t'|_{p'} = v$, $f$ is associative, $depth(p') = depth(p) + 1$, and $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$; or*
4. *$u = f(u_1, \ldots, u_k)$, $t|_p = f(x_1, \ldots, x_{m_1}, u_1, \ldots, u_k)$, $v = f(v_1, \ldots, v_n)$, $t'|_{p'} = f(w_1, \ldots, w_{m_2}, v_1, \ldots, v_n)$, $f$ is associative, $depth(p) = depth(p')$, and $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$.*

**Proof.** Straightforward by successive application of the inference rule Decompose of Figure 1.1 and the inference rule $\text{Decompose}_A$ of Figure 3.3.

■

**Definition 3.4.7 (Associative Conflict Pair)** *Given flattened terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative, the pair $(u, v)$ is called an* associative conflict pair of $t$ and $t'$ *iff there exist at*

*least one position $p \in Pos(t)$ and at least one position $p' \in Pos(t')$ such that either:*

1. $t|_p = u$, $t'|_{p'} = v$, $u \neq_E v$, $depth(p) = depth(p')$, *and* $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$; *or*

2. $t|_p = u$, $v = f(v_1, \ldots, v_n)$, $t'|_{p'} = f(w_1, \ldots, w_m, v_1, \ldots, v_n, w'_1, \ldots, w'_{m'})$, $f$ *is associative,* $root(u) \neq f$, $depth(p) = depth(p') + 1$, *and* $\forall 1 \leq i \leq depth(p')$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$; *or*

3. $u = f(u_1, \ldots, u_n)$, $t|_p = f(w_1, \ldots, w_m, u_1, \ldots, u_n, w'_1, \ldots, w'_{m'})$, $t'|_{p'} = v$, $f$ *is associative,* $root(v) \neq f$, $depth(p') = depth(p) + 1$, *and* $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$.

Note that the least general generalization 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 generalization 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 $f(e, e, e)$.

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

**Lemma 3.4.8** *Given flattened terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative, and a fresh variable $x$, $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle CT \mid u \stackrel{y}{\triangleq} v \wedge S \mid \theta \rangle$ using the inference rules of Figures 3.1 and 3.3 iff $(u, v)$ is an associative conflict pair of $t$ and $t'$.*

**Proof.** ($\Rightarrow$) If $u \stackrel{y}{\triangleq} v \in S$, then there must be two configurations $\langle u \stackrel{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1 \rangle$, $\langle CT_2 \mid u \stackrel{y}{\triangleq} v \wedge S_2 \mid \theta_2 \rangle$ such that

$$\begin{array}{rl} & \langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \\ \rightarrow^* & \langle u \stackrel{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1 \rangle \\ \rightarrow & \langle CT_2 \mid u \stackrel{y}{\triangleq} v \wedge S_2 \mid \theta_2 \rangle \end{array}$$

$$\rightarrow^* \qquad \langle \emptyset \mid S \mid \theta \rangle,$$

$u \stackrel{y}{\triangleq} v \notin S_1$, $u \stackrel{y}{\triangleq} v \notin CT_2$, and $root(u) \neq root(v)$. By Lemma 3.4.6, there exist a position $p \in Pos(t)$ and a position $p' \in Pos(t')$ such that either:

1. $t|_p = u$, $t'|_{p'} = v$, $depth(p) = depth(p')$, and $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$; or
2. $t|_p = u$, $v = f(v_1, \ldots, v_n)$, $t'|_{p'} = f(w_1, \ldots, w_m, v_1, \ldots, v_n, w'_1, \ldots, w'_{m'})$, $f$ is associative, $depth(p) = depth(p') + 1$, and $\forall 1 \leq i \leq depth(p')$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$; or
3. $u = f(u_1, \ldots, u_n)$, $t|_p = f(w_1, \ldots, w_m, u_1, \ldots, u_n, w'_1, \ldots, w'_{m'})$, $t'|_{p'} = v$, $f$ is associative, $depth(p') = depth(p) + 1$, and $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$.

Note that, since $root(u) \neq root(v)$, the fourth case of Lemma 3.4.6 is not possible. Therefore, either $(u, v)$ (or $(u, f(v_1, \ldots, v_n))$) is an associative conflict pair.

($\Leftarrow$) By Lemma 3.4.6, there is a configuration $\langle u \stackrel{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1 \rangle$ such that $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle u \stackrel{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1 \rangle$, $u \stackrel{y}{\triangleq} v \notin S_1$, and $root(u) \neq root(v)$. Then, the inference rule Solve is applied, i.e., $\langle u \stackrel{y}{\triangleq} v \wedge CT_1 \mid S_1 \mid \theta_1 \rangle \rightarrow \langle CT_1 \mid u \stackrel{y}{\triangleq} v \wedge S_1 \mid \theta_1 \rangle$ and $u \stackrel{y}{\triangleq} v$ will be part of $S$ in the final configuration $\langle \emptyset \mid S \mid \theta \rangle$.

■

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

**Lemma 3.4.9** *Given flattened terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative, and a fresh variable $x$, $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle C \mid S \mid \theta \rangle$ using the inference rules of Figures 3.1 and 3.3 iff $x\theta$ is a generalization of $t$ and $t'$ modulo associativity.*

**Proof.** By structural induction on the term $x\theta$ (or $u$). If $x\theta = x$, then $\theta = id$ and the conclusion follows. If $x\theta = f(u_1, \ldots, u_k)$ and $f$ is free, then the inference rule Decompose$_E$ of Figure 3.1 is applied and we have

that $t = f(t_1, \ldots, t_k)$ and $t' = f(t'_1, \ldots, t'_k)$. If $x\theta = f(u_1, \ldots, u_k)$ and $f$ is associative, then the inference rule Decompose$_A$ of Figure 3.3 is applied and we have that $t = f(u_1, \ldots, u_n)$, $t' = f(v_1, \ldots, v_m)$, and $k = min(n, m)$. Let us consider the different values for $k$, $n$ and $m$.

- If $k = n = m$, then by induction hypothesis $t_i$ is a generalization of $u_i$ and $v_i$, for each $i \in \{1, \ldots, k\}$. Now, if there are no shared variables among all $t_i$, then the conclusion follows. Otherwise, for each variable $z$ shared between two different terms $t_i$ and $t_j$, there is a constraint $w_1 \overset{z}{\triangleq} w_2 \in S$ and, by Lemma 3.4.8, there is a conflict pair $(w_1, w_2)$ in $t_i$ and $t'_i$. Thus, the conclusion follows.

- If $k = n$ and $\ell = m - n$, then there is an element $j \in \{1, \ldots, k\}$ such that by induction hypothesis $t_i$ is a generalization of $u_i$ and $v_i$ for each $i < j$, $t_j$ is a generalization of $u_j$ and $f(v_j, \ldots, v_{j+\ell})$, and $t_i$ is a generalization of $u_i$ and $v_{i+\ell}$ for each $i > j$. Now, if there are no shared variables among all $t_i$ s.t. $i \neq j$, then the conclusion follows. Otherwise, for each variable $z$ shared between two different terms $t_{i_1}$ and $t_{i_2}$ s.t. $i_1 \neq j$ and $i_2 \neq j$, there is a constraint $w_1 \overset{z}{\triangleq} w_2 \in S$ and, by Lemma 3.4.8, there is a 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 3.4.10 (Correctness and Completeness)** *Given an equational theory $(\Sigma, E)$, and flattened $\Sigma$-terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative, and a fresh variable $x$, then $u \in gen_E(t, t')$ iff there is $u'$ in $\{x\theta \mid \langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle\}$ using the inference rules of Figures 3.1 and 3.3 such that $u \simeq_E u'$.*

**Proof.** Similar to Theorem 3.3.9.

■

Recall that the inference rules of Figures 3.1 and Figure 3.3 together are not confluent, so that 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 generalizations $x\theta_1, \ldots, x\theta_n$.

**Decompose**$_{AC}$

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

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

Figure 3.5: Decomposition rule for an associative–commutative function symbol $f$

## 3.5 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 3.4. 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 (Hullot, 1980; Eker, 2003) such that $=_{AC}$ is decidable.

The new decomposition rule for the AC case is similar to the decompose inference rule for associative function symbols, except that all permutations of $f(t_1, \ldots, t_n)$ and $f(s_1, \ldots, s_m)$ are considered. As before, the AC generalization of $t$ and $s$ are the maximal elements w.r.t. $<_{AC}$ of the normal forms of $t \overset{x}{\triangleq} t'$ w.r.t. the new extended generalization calculus. Just notice that this rule is (don't know) non-deterministic, hence all possibilities must be explored.

Similarly to the rule *Decompose*$_A$, we give the rule *Decompose*$_{AC}$ for the case when, in the generalization problem $f(t_1, ..., t_n) \overset{x}{\triangleq} f(s_1, ..., s_m)$, we have that $n \geq m$. For the other way around, i.e., $n < m$, a similar rule would be needed, that we omit since it is entirely similar. To simplify, we write $\{i_1, \ldots, i_k\} \uplus \{i_{k+1}, \ldots, i_n\} = \{1, \ldots, n\}$ to denote that the sequence $\{i_1, \ldots, i_n\}$ is a permutation of the sequence $\{1, \ldots, n\}$ and, given an element $k \in \{1, \ldots, n\}$, we split the sequence $\{i_1, \ldots, i_n\}$ in the two parts, $\{i_1, \ldots, i_k\}$ and $\{i_{k+1}, \ldots, i_n\}$.

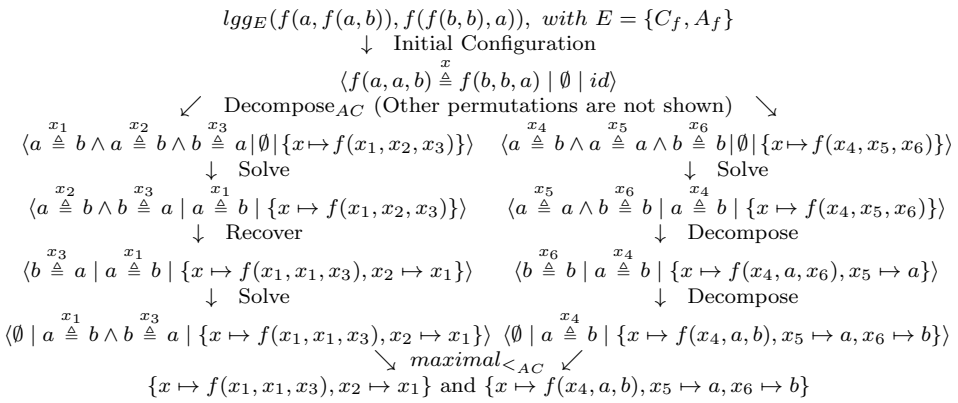


Figure 3.6: Computation trace for the AC–generalizations of terms $f(a, f(a, b))$ and $f(f(b, b), a)$.

**Example 3.5.1**

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., $ax(f) = \{A_f, C_f\}$. By applying the rules $Solve_E$, $Recover_E$, and $Decompose_{AC}$ above, we end in two terminal configurations whose respective substitution components are $\theta_1 = \{x \mapsto f(x_1, x_1, x_3), x_2 \mapsto x_1\}$ and $\theta_2 = \{x \mapsto f(x_4, a, b), x_5 \mapsto a, x_6 \mapsto b\}$, thus we compute that the lggs modulo $AC$ of $t$ and $s$ are $f(x_1, x_1, x_3)$ and $f(x_4, a, b)$. The corresponding computation trace is shown in Figure 3.6.

Termination is straightforward.

**Theorem 3.5.2 (Termination)** *Given an equational theory $(\Sigma, E)$, $\Sigma$-terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative-commutative, and $x$ is a variable, every derivation stemming from an initial configuration $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle$ using the inference rules of Figures 3.1 and 3.5 terminates with a configuration $\langle \emptyset \mid S \mid \theta\rangle$.*

**Proof.** Similar to the proof of Theorem 1.1.1. ∎

In order to prove correctness and completeness of the lgg calculus modulo AC, similarly to Definitions 1.2.3, 3.3.6, and 3.4.7, we introduce the auxiliary concept of an *associative-commutative conflict pair*, and prove the appropriate auxiliary results.

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

**Lemma 3.5.3 (Uniqueness of Generalization Variables)** *Lemma 1.1.2 holds for $t \overset{x}{\triangleq} t'$ when the symbols in $t$, $t'$ are free or associative-commutative, for the inference rules of Figures 3.1 and 3.5.*

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.

**Lemma 3.5.4** *Given terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative-commutative, and a fresh variable $x$ such that $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle CT \mid S \mid \theta\rangle$ using the inference rules of Figures 3.1 and 3.5, then $Index(S \cup CT) \subseteq Ran(\theta)$, and $Ran(\theta) = Var(x\theta)$.*

**Proof.** Immediate by construction.

■

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

**Lemma 3.5.5** *Given flattened terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative-commutative, and a fresh variable $x$, $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle u \overset{y}{\triangleq} v \wedge CT \mid S \mid \theta\rangle$ using the inference rules of Figures 3.1 and 3.5 iff there exist a position $p \in Pos(t)$ and a position $p' \in Pos(t')$ such that either:*

1. *$t|_p = u$, $t'|_{p'} = v$, $depth(p) = depth(p')$, and $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$; or*

2. *$t|_p = u$, $v = f(v_1, \ldots, v_n)$, $t'|_{p'} = f(w_1, \ldots, w_m)$, $f$ is associative-commutative, for each $i \in \{1, \ldots, n\}$ there is $j \in \{1, \ldots, m\}$ s.t. $v_i =_E w_j$, $depth(p) = depth(p') + 1$, and $\forall 1 \leq i \leq depth(p')$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$; or*

3. $u = f(u_1, \ldots, u_n)$, $t|_p = f(w_1, \ldots, w_m)$, $t'|_{p'} = v$, $f$ *is associative-commutative, for each* $j \in \{1, \ldots, m\}$ *there is* $i \in \{1, \ldots, n\}$ *s.t.* $w_j =_E v_i$, $depth(p') = depth(p) + 1$, *and* $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$.

**Proof.** Straightforward by successive application of the inference rule Decompose of Figure 1.1 and the inference rule $\text{Decompose}_{AC}$ of Figure 3.5.

■

**Definition 3.5.6 (Associative-commutative Conflict Pair)** *Given flattened terms* $t$ *and* $t'$ *such that every symbol in* $t$ *and* $t'$ *is free or associativ-com mutative, the pair* $(u, v)$ *is called an* associative conflict pair of $t$ and $t'$ *iff there exist at least one position* $p \in Pos(t)$ *and at least one position* $p' \in Pos(t')$ *such that either:*

1. $t|_p = u$, $t'|_{p'} = v$, $u \neq_E v$, $depth(p) = depth(p')$, *and* $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$; *or*

2. $t|_p = u$, $v = f(v_1, \ldots, v_n)$, $t'|_{p'} = f(w_1, \ldots, w_m)$, $f$ *is associative-commutative, for each* $i \in \{1, \ldots, n\}$ *there is* $j \in \{1, \ldots, m\}$ *s.t.* $v_i =_E w_j$, $u \neq_E v$, $depth(p) = depth(p') + 1$, *and* $\forall 1 \leq i \leq depth(p')$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$; *or*

3. $u = f(u_1, \ldots, u_n)$, $t|_p = f(w_1, \ldots, w_m)$, $t'|_{p'} = v$, $f$ *is associative-commutative, for each* $j \in \{1, \ldots, m\}$ *there is* $i \in \{1, \ldots, n\}$ *s.t.* $w_j =_E v_i$, $u \neq_E v$, $depth(p') = depth(p)+1$, *and* $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$.

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

**Lemma 3.5.7** *Given flattened terms* $t$ *and* $t'$ *such that every symbol in* $t$ *and* $t'$ *is free or associative-commutative, and a fresh variable* $x$, $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle CT \mid u \stackrel{y}{\triangleq} v \wedge S \mid \theta \rangle$ *using the inference rules of Figures 3.1 and 3.5 iff* $(u, v)$ *is an associative-commutative conflict pair of* $t$ *and* $t'$.

**Proof.** Similar to the proof of Lemma 3.4.8 but using Lemma 3.5.5 instead of Lemma 3.4.6 and Definition 3.5.6 instead of Definition 3.4.7. ■

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

**Lemma 3.5.8** *Given flattened terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative-commutative, and a fresh variable $x$, $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figures 3.1 and 3.5 iff $x\theta$ is a generalization of $t$ and $t'$ modulo associativity-commutativity.*

**Proof.** Similar to the proof of Lemma 3.4.9 but using Lemma 3.5.7 instead of Lemma 3.4.8 and Definition 3.5.6 instead of Definition 3.4.7. ■

Finally, correctness and completeness are proved as follows.

**Theorem 3.5.9 (Correctness and Completeness)** *Given an equational theory $(\Sigma, E)$, flattened $\Sigma$-terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or associative-commutative, and a fresh variable $x$, then $u \in gen_E(t, t')$ iff there is $u'$ in $\{x\theta \mid \langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle\}$ using the inference rules of Figures 3.1 and 3.5 such that $u \simeq_E u'$.*

**Proof.** Similar to Theorem 3.3.9. ■

Recall that the inference rules of Figures 3.1 and 3.5 together are not confluent, so that 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 generalizations $x\theta_1, \ldots, x\theta_n$.

## 3.6 Least general generalization modulo $U$

Finally, let us introduce the inference rule of Figure 3.7 for handling function symbols $f$ which have an identity element $e$. This rule considers the identity axioms in a rather lazy or on-demand manner 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

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

Figure 3.7: Inference rule for expanding function symbol $f$ with identity element $e$

side of the constraint $t \stackrel{x}{\triangleq} s$ 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 that is entirely similar.

**Example 3.6.1**
Let $t = f(a,b,c,d)$ and $s = f(a,c)$ be two terms where $ax(f) = \{A_f, C_f, U_f\}$. By applying the rules $Solve_E$, $Recover_E$, $Decompose_{AC}$, and $Expand_U$ above, we end in a terminal configuration $\langle \emptyset \mid S \mid \theta\rangle$, where $\theta = \{x \mapsto f(a, f(c, f(x_5, x_6))),\ x_1 \mapsto a, x_2 \mapsto f(c, f(x_5, x_6)), x_3 \mapsto c, x_4 \mapsto f(x_5, x_6)\}$, thus we compute that the lgg modulo $ACU$ of $t$ and $s$ is $f(a,c,x_5,x_6)$. The computation trace is shown in Figure 3.8.

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 3.6.2**
Let $t = f(f(a,a), f(b,a))$ and $t' = f(f(b,b), a)$ be two terms such that $\{A_f, U_f\} \subseteq ax(f)$. 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 are more general terms.

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

**Theorem 3.6.3 (Termination)** *Given an equational theory $(\Sigma, E)$, $\Sigma$-terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or with identity element $e$, and a fresh variable $x$, every derivation stemming from an*

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

Figure 3.8: Computation trace for the ACU–generalization of terms $f(a,b,c,d)$ and $f(a,c)$.

*initial configuration* $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle$ *using the inference rules of Figures 3.1 and 3.7 terminates with a configuration* $\langle \emptyset \mid S \mid \theta \rangle$.

**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 $\text{Decompose}_E$ of Figure 3.1 can be applied. Let $\overline{k}$ be the maximum of $|t|$ and $|t'|$. Since the inference rule $\text{Expand}_U$ 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 $\text{Expand}_U$ followed by a decomposing rule of Figure 3.1 (or Figures 3.2, 3.3, and 3.5) can be applied. Finally, the application of rules $\text{Solve}_E$ and $\text{Recover}_E$ strictly decreases the size $|CT|$ of the $CT$ component of the lgg configurations

at each step, hence the derivation terminates.

■

In order to prove correctness and completeness, we introduce the auxiliary concepts of an identity conflict pair, similarly to Definitions 1.2.3, 3.3.6, 3.4.7, and 3.5.6, plus some auxiliary results.

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

**Lemma 3.6.4 (Uniqueness of Generalization Variables)** *Lemma 1.1.2 holds for $t \overset{x}{\triangleq} t'$ when the symbols in $t$, $t'$ are free or with identity element $e$, for the inference rules of Figures 3.1 and 3.7.*

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.

**Lemma 3.6.5** *Given terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or with identity element $e$, and a fresh variable $x$ such that $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle CT \mid S \mid \theta\rangle$ using the inference rules of Figures 3.1 and 3.7, then $Index(S \cup CT) \subseteq Ran(\theta)$, and $Ran(\theta) = Var(x\theta)$.*

**Proof.** Immediate by construction.

■

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

**Lemma 3.6.6** *Given terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or with identity element $e$, and a fresh variable $x$, then $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle u \overset{y}{\triangleq} v \wedge CT \mid S \mid \theta\rangle$ using the inference rules of Figures 3.1 and 3.7 iff there exist a position $p \in Pos(t)$ and a position $p' \in Pos(t')$ such that either:*

1. *$t|_p = u$, $t'|_{p'} = v$, $\forall 1 \leq i \leq min(depth(p), depth(p'))$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$, and*

- *if* $p \geq p'$, *then* $\forall i \in \{depth(p'), \ldots, depth(p)\}$, $root(t|_{p|_i}) = f$ *s.t.* $f$ *has identity element* $e$, *or*
- *if* $p' > p$, *then* $\forall i \in \{depth(p), \ldots, depth(p')\}$, $root(t'|_{p'|_i}) = f$ *s.t.* $f$ *has identity element* $e$; *or*

2. $t|_p = u$, $v = e$, $p' = p|_{depth(p)-1}$, $root(t|_{p'}) = f$ *s.t.* $f$ *has identity element* $e$, *and* $\forall 1 \leq i \leq depth(p')$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$; *or*

3. $u = e$, $t'|_{p'} = v$, $p = p'|_{depth(p')-1}$, $root(t'|_p) = f$ *s.t.* $f$ *has identity element* $e$, *and* $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$.

**Proof.** Straightforward by successive application of the inference rule Decompose of Figure 1.1 and the inference rule $\text{Decompose}_U$ of Figure 3.7.

■

**Definition 3.6.7 (Identity Conflict Pair)** *Given terms* $t$ *and* $t'$ *such that every symbol in* $t$ *and* $t'$ *is free or with identity element* $e$, *the pair* $(u, v)$ *is called an* identity conflict pair of $t$ and $t'$ *iff there exist at least one position* $p \in Pos(t)$ *and at least one position* $p' \in Pos(t')$ *such that either:*

1. $t|_p = u$, $t'|_{p'} = v$, $u \neq_E v$, $\forall 1 \leq i \leq min(depth(p), depth(p'))$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$, *and*
   - *if* $p \geq p'$, *then* $\forall i \in \{depth(p'), \ldots, depth(p)\}$, $root(t|_{p|_i}) = f$ *s.t.* $f$ *has identity element* $e$, *or*
   - *if* $p' > p$, *then* $\forall i \in \{depth(p), \ldots, depth(p')\}$, $root(t'|_{p'|_i}) = f$ *s.t.* $f$ *has identity element* $e$; *or*
2. $t|_p = u$, $v = e$, $u \neq_E e$, $p' = p|_{depth(p)-1}$, $root(t|_{p'}) = f$ *s.t.* $f$ *has identity element* $e$, *and* $\forall 1 \leq i \leq depth(p')$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$; *or*
3. $u = e$, $t'|_{p'} = v$, $v \neq_E e$, $p = p'|_{depth(p')-1}$, $root(t'|_p) = f$ *s.t.* $f$ *has identity element* $e$, *and* $\forall 1 \leq i \leq depth(p)$, $root(t|_{p|_i}) = root(t'|_{p'|_i})$.

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

**Lemma 3.6.8** *Given terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or has an identity element, and a fresh variable $x$, $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle CT \mid u \overset{y}{\triangleq} v \wedge S \mid \theta\rangle$ using the inference rules of Figures 3.1 and 3.7 iff $(u, v)$ is an identity conflict pair of $t$ and $t'$.*

**Proof.** Similar to the proof of Lemma 3.4.8 but using Lemma 3.6.6 instead of Lemma 3.4.6 and Definition 3.6.7 instead of Definition 3.4.7. ■

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

**Lemma 3.6.9** *Given terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or has an identity element, and a fresh variable $x$, $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$, using the inference rules of Figures 3.1 and 3.7 iff $x\theta$ is a generalization of $t$ and $t'$ modulo identity.*

**Proof.** Similar to the proof of Lemma 3.4.9 but using Lemma 3.6.8 instead of Lemma 3.4.8 and Definition 3.6.7 instead of Definition 3.4.7. ■

Finally, correctness and completeness are proved as follows.

**Theorem 3.6.10 (Correctness and Completeness)** *Given an equational theory $(\Sigma, E)$, $\Sigma$-terms $t$ and $t'$ such that every symbol in $t$ and $t'$ is free or has an identity element, and a fresh variable $x$, then $u \in gen_E(t, t')$ iff there is $u'$ in $\{x\theta \mid \langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle\}$ using the inference rules of Figures 3.1 and 3.7 such that $u \simeq_E u'$.*

**Proof.** Similar to Theorem 3.3.9. ■

Recall that the inference rules of Figures 3.1 and Figure 3.7 together are not confluent, hence 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 generalizations $x\theta_1, \ldots, x\theta_n$. Note that if the 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 3.7.

## 3.7 A general 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 3.1, 3.2, 3.3, 3.5, and 3.7) 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 $\Sigma$ 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 3.7.1 (Termination)** *For an equational theory $(\Sigma, E)$ with $E \in \mathbb{E}$, two $\Sigma$-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 3.1, 3.2, 3.3, 3.5, and 3.7 terminates with a configuration $\langle \emptyset \mid S \mid \theta \rangle$.*

The correctness and completeness of our algorithm is ensured by:

**Theorem 3.7.2 (Correctness and Completeness)** *Given an equational theory $(\Sigma, E)$ with $E \in \mathbb{E}$, $\Sigma$-terms $t$ and $t'$, and a fresh variable $x$, then $u \in gen_E(t, t')$ iff there is $u'$ in $\{x\theta \mid \langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id \rangle \rightarrow^* \langle \emptyset \mid S \mid \theta \rangle\}$ using the inference rules of Figures 3.1, 3.2, 3.3, 3.5, and 3.7 such that $u \simeq_E u'$.*

# 4

# Order-Sorted Least General Generalizations modulo E

In this chapter, we generalize the unsorted modular equational generalization algorithm presented in Chapter 3 to the order-sorted setting.

First of all, we assume that a kind-completed, pre-regular, order-sorted signature $(\Sigma, \mathsf{S}, <)$ has the same equational attributes for overloaded symbols, i.e., for any two operator declarations of symbol $f$ with arity $n$, $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$ and $f : \mathsf{s}'_1 \times \ldots \times \mathsf{s}'_n \to \mathsf{s}'$ such that $\mathsf{s}_i \leq \mathsf{s}_i$ for $1 \leq i \leq n$, if an equation $t = t'$ is applicable to $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$, it must also be applicable to $f : \mathsf{s}'_1 \times \ldots \times \mathsf{s}'_n \to \mathsf{s}'$.

As in Chapter 2, we consider two terms $t$ and $t'$ having the same top sort, otherwise they are incomparable and no generalization 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 Chapter 3, when different function symbols satisfying different associativity and/or commutativity and/or identity axioms are considered, we can use the inference rules of Figures 4.1, 4.2, 4.3, 4.4, and 4.5 all together.

Note that we have just followed the same approach of Chapter 2 and extended the inference rules of Figures 3.1, 3.2, 3.3, 3.5, and 3.7 to Figures 4.1, 4.2, 4.3, 4.4, and 4.5 provided below.

## 4.1 Termination

Termination is straightforward.

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

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 $n \geq 0$.

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

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

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

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

Figure 4.1: Basic inference rules for least general $E$–generalization

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

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 4.2: Decomposition rule for a commutative function symbol $f$

**Decompose**$_A$

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

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 4.3: Decomposition rule for an associative (non–commutative) function symbol $f$

**Decompose**$_{AC}$

$$\frac{f : [\mathsf{s}] \times [\mathsf{s}] \to [\mathsf{s}] \wedge \{A_f, C_f\} \subseteq ax(f) \wedge n \geq m \wedge \{i_1, \ldots, i_{m-1}\} \uplus \{i_m, \ldots, i_n\} = \{1, \ldots, n\}}{\begin{array}{l}\langle f(t_1, \ldots, t_n) \stackrel{x:[\mathsf{s}]}{\triangleq} f(t'_1, \ldots, t'_m) \wedge C \mid S \mid \theta\rangle \\ \to \langle t_{i_1} \stackrel{t'_1:[\mathsf{s}]}{\triangleq} t'_1 \wedge \ldots \wedge t_{i_{m-1}} \stackrel{x_{m-1}:[\mathsf{s}]}{\triangleq} t'_{m-1} \wedge f(t_{i_m}, \ldots, t_{i_n}) \stackrel{x_m:[\mathsf{s}]}{\triangleq} t'_m \wedge C \mid S \mid \theta\sigma\rangle\end{array}}$$

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

Figure 4.4: Decomposition rule for an associative–commutative function symbol $f$

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

Figure 4.5: Inference rule for expanding function symbol $f$ with identity element $e$

**Theorem 4.1.1 (Termination)** *Given a kind-completed, pre-regular, order-sorted equational theory $(\Sigma, E)$ with the same equational attributes for overloaded symbols, 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 4.1, 4.2, 4.3, 4.4, and 4.5 terminates with a configuration $\langle \emptyset \mid S \mid \theta \rangle$.*

**Proof.** Similar to the proofs of Theorems 1.1.1 and 3.6.3.

■

## 4.2 Correctness and Completeness

In order to prove correctness and completeness, Definitions 3.3.6, 3.4.7, 3.5.6, and 3.6.7 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 2.2 and define order-sorted $E$-lgg computation by subsort specialization. That is, to compute generalizations by removing sorts (i.e., upgrading variables to top sorts), computing (unsorted) $E$-lggs, and then 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.

**Definition 4.2.1 (A-Subterm Replacement)** *Given two flattened terms $t$ and $t'$ and an associative conflict pair $(u, v)$ with conflict positions $p \in Pos(t)$ and $p' \in Pos(t')$ such that $t|_p = u$, $v = f(v_1, \ldots, v_n)$, $t'|_{p'} = f(w_1, \ldots, w_m, v_1, \ldots, v_n, w'_1, \ldots, w'_{m'})$, and $f$ is associative, we write $t[[x{:}\mathsf{s}]]_p$ and $t'[[x{:}\mathsf{s}]]_{p'}$ to denote the terms $t[[x{:}\mathsf{s}]]_p = t[x{:}\mathsf{s}]_p$ and $t'[[x{:}\mathsf{s}]]_{p'} = t'[f(w_1, \ldots, w_m, x{:}\mathsf{s}, w'_1, \ldots, w'_{m'})]_{p'}$.*

**Definition 4.2.2 (AC-Subterm Replacement)** *Given two flattened terms $t$ and $t'$ and an associative-commutative conflict pair $(u, v)$ with conflict positions $p \in Pos(t)$ and $p' \in Pos(t')$ such that $t|_p = u$, $v =$*

*$f(v_1, \ldots, v_n)$, $t'|_{p'} = f(w_1, \ldots, w_m)$, $f$ is associative-commutative, for each $i \in \{1, \ldots, n\}$ there is $j \in \{1, \ldots, m\}$ s.t. $v_i =_E w_j$, we write $t[[x{:}\mathsf{s}]]_p$ and $t'[[x{:}\mathsf{s}]]_{p'}$ to denote the terms $t[[x{:}\mathsf{s}]]_p = t[x{:}\mathsf{s}]_p$ and $t'[[x{:}\mathsf{s}]]_{p'} = t'[f(w'_1, \ldots, w'_k, x{:}\mathsf{s})]_{p'}$ where $\{w'_1, \ldots, w'_k\} = \{w \in \{w_1, \ldots, w_m\} \mid \nexists i \in \{1, \ldots, n\}, w =_E v_i\}$.*

As in Section 2.2, we define order-sorted $E$-lgg computation by subsort specialization using a top-sorted generalization (see Definition 2.2.1) and a sort-specialized generalization (see Definition 4.2.4).

**Definition 4.2.3 (Top-sorted Equational Generalization)** *Given a kind-completed, pre-regular, order-sorted equational theory $(\Sigma, E)$ with the same equational attributes for overloaded symbols, and flattened $\Sigma$-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 $(p^i_1, \ldots, p^i_{n_i}, q^i_1, \ldots, q^i_{n_i})$ 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*

$$tsg_E(t, t') = ((t[[x_1{:}\mathsf{s_1}]]_{p^1_1, \ldots, p^1_{n_1}}) \cdots)[[x_k{:}\mathsf{s_k}]]_{p^k_1, \ldots, p^k_{n_k}}$$

*where $x_1{:}\mathsf{s_1}, \ldots, x_k{:}\mathsf{s_k}$ are fresh variables.*

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

**Definition 4.2.4 (Sort-specialized Equational Generalization)** *Given a kind-completed, pre-regular, order-sorted equational theory $(\Sigma, E)$ with the same equational attributes for overloaded symbols, and flattened $\Sigma$-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 $t'$. We define*

$$\begin{aligned} \textit{sort-down-subs}_E(t, t') = \{\rho \mid\ & Dom(\rho) = \{x_1{:}\mathsf{s_1}, \ldots, x_k{:}\mathsf{s_k}\} \\ & \wedge\ \forall 1 \leq i \leq k, \rho(x_i{:}\mathsf{s_i}) = x_i{:}\mathsf{s'_i} \\ & \wedge\ \mathsf{s'_i} \in LUBS(LS(u_i), LS(v_i))\} \end{aligned}$$

*where all the $x_i{:}\mathsf{s'_i}$ are fresh variables. The set of sort-specialized $E$-generalizations 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 4.2.5** *Given a kind-completed, pre-regular, order-sorted equational theory $(\Sigma, E)$ with the same equational attributes for overloaded symbols, and flattened $\Sigma$-terms $t$ and $t'$ such that $[LS(t)] = [LS(t')]$, $tsg_E(t, t')$ is a order-sorted equational generalization 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 2.2.5.

■

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

**Theorem 4.2.6 (Correctness and Completeness)** *Given a kind-completed,*
*pre-regular, order-sorted equational theory $(\Sigma, E)$ with the same equational attributes for overloaded symbols, flattened $\Sigma$-terms $t$ and $t'$ such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, and a fresh variable $x{:}[\mathsf{s}]$, $u \in lgg_E(t, t')$ is an order-sorted equational lgg of $t$ and $t'$ iff $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ using the inference rules of Figures 4.1, 4.2, 4.3, 4.4, and 4.5 for some $S$ and $\theta$ and $u \simeq_E (x{:}[\mathsf{s}])\theta$.*

**Proof.** Similar to the proof of Theorem 2.3.4.

■

# 5

# The ACUOS System

This chapter presents ACUOS, an optimized implementation of the order-sorted modular ACU least general generalization algorithm that we formalized in Chapter 4. The tool consists of about a thousand lines of code written in the high-performance Maude equational programming language (Meseguer, 1992), taking advantage of its powerful reflective capabilities to express inference rules and encode terms.

The ACUOS system integrates three different front-ends: (*i*) a Maude meta-level function, which provides direct access to the complete back-end functionality; (*ii*) a Full Maude user level command, which liberates the user from ancillary low-level technicalities; and (*iii*) a Web interface that enables the use of the tool directly from a Web browser. Both the application source code and the ACUOS Web interface are publicly available at `http://safe-tools.dsic.upv.es/acuos`.

## 5.1 Algorithmic improvements

In order to optimize the tool performance, we have introduced a number of significant improvements w.r.t. the original algorithm of (Alpuente et al., 2012). First, the inference rules have been carefully encoded as a memoization-based, recursive procedure that does not carry any configuration *store* and avoids unnecessary re-computations. We also identified the inference rules that are confluent and encoded them as Maude equations (instead of rules), highly reducing the search space as well as the memory usage due to the different treatment of rules and equations in Maude (Clavel et al., 2007). Finally, unlike the original algorithm, the *sort* information is not incrementally computed but is more efficiently

introduced into the solution by a suitable and inexpensive sorting post-processing.

## 5.2 Architecture

The tool takes as input two input structures (terms) and a Maude module that specifies the signature for the function symbols involved. The architecture of ACUOS, which is depicted in Figure 5.1, comprises the following modules:

**(i) Pre-processing.** The input Maude specification and the input terms are lifted to the meta-level. Using their meta-level representation, an initial configuration for the generalization procedure is built that is lifted to the meta-level again so that Maude's meta-level search operators can be used to find all ACU generalizers.

**(ii) Search for candidates.** Maude's `metaSearch` function is used to compute the final configurations of the recursive generalization procedure.

**(iii) Order-sorting addition.** Unresolved conflict pairs are substituted by fresh variables. If there is more than one possible sort for a given variable, each of them gives rise to a different candidate.

**(iv) Normalization.** A deterministic variable renaming algorithm that reduces term equality modulo renaming to syntactic equality is applied. As a result, a complete and finite (although not minimal) set of valid generalizers is delivered.

**(v) Minimization post-processing.** The candidate set is minimized according to the instantiation ordering, producing the set of least general, order-sorted ACU generalizers of the input terms.

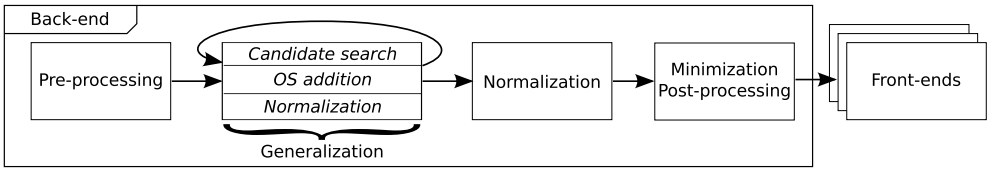


Figure 5.1: Architecture of the ACUOS system

## 5.3 Interface

ACUOS comes with an intuitive Web user interface based on the AJAX technology, which allows the tool to be used through a Java Web application (see Appendix A for details). The back-end of the tool has been implemented as a Maude meta-level function named `metaGeneralize`, which computes the set of generalizers of two terms in a given signature. For convenience, we also provide a *Full Maude* (Clavel et al., 2007) extension that offers a user-level command (`get lggs`), allowing the user to harness the full power of the tool while being liberated from ancillary meta-level technicalities.

For instance, consider the following Full Maude[1] module:

```
(mod fACU-OS is
    sorts E A B C D Empty .

    subsort Empty < A B .

    subsort A < C D .
    subsort B < C D .

    subsort C < E .
    subsort D < E .

    op a : ->  A .
    op b : ->  B .
    op c : ->  C .
    op d : ->  D .
    op e : ->  Empty .

    op f : Empty Empty -> Empty [assoc comm id: e] .
```

[1] We refer the reader to (Clavel et al., 2007) for Maude and Full Maude syntax.

```
    op f : A A -> A [assoc comm id: e] .
    op f : B B -> B [assoc comm id: e] .
    op f : E E -> E [assoc comm id: e] .
endm)
```

This module is automatically extended to its kind-complete version by Maude. It defines five constants `a`, `b`, `c`, `d`, `e` and four binary symbols sharing the same name `f` but with different signatures corresponding to the subsort structure of Figure 2.3. All four versions of symbol $f$ (plus its kind extension $f : [\mathsf{E}] \to [\mathsf{E}]$) are associative-commutative and with identity symbol the constant $e$. Now, we can type the following generalization problem in Full Maude obtaining the six possible order-sorted $E$-lggs.

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

Lgg 1
f(X1:C,b,a)

Lgg 2
f(X1:D,b,a)

Lgg 3
f(X1:C,X1:C,X3:C)

Lgg 4
f(X1:C,X1:C,X3:D)

Lgg 5
f(X1:D,X1:D,X3:C)

Lgg 6
f(X1:D,X1:D,X3:D)

No more lgg.
```

## 5.4 Generalizing data structures

As mentioned above, the generalization problem for two terms $t_1$ and $t_2$ consists in finding their *least general generalization* (lgg), i.e., the least general term $t$ such that both $t_1$ and $t_2$ are instances of $t$ under appropriate substitutions. For instance, the term `siblings(X,Y)`

is a generalization of `siblings(john,sam)` and `siblings(tom,sam)`, but their least general generalizer is `siblings(X,sam)`. We also recall that generalizer terms are not always linear: the generalization of `siblings(john,john,sam)` and `siblings(tom,tom,sam)` is `siblings(X,X,sam)`, not `siblings(X,Y,sam)`.

While ordinary, syntactic generalization is useful for some applications, it suffers two important limitations. First, it cannot generalize common data structures such as records, lists, sets, and multisets. For instance, given a record structure $r(field_1 : v_1, field_2 : v_2, \ldots, field_n : v_n)$, we want to recognize the tuple $(field_1 : v_1, ..., field_n : v_n)$ of the record fields irrespectively of the ordering of the elements; or more formally, two record structures are equal if they are equal *modulo the commutativity and associativity* of the tuple constructor $(,)$. The problem is similar for lists (concatenation is associative with unity *nil*), multisets (insertion is associative-commutative with unity $\emptyset$), or sets (insertion is associative-commutative-idempotent with unity $\emptyset$).

Let us introduce the constants `john`, `sam`, `peter`, `tom`, and `mary` and consider a relation symbol `siblings` whose only argument is a tuple of such constants that we build by using the tuple constructor $(,)$ as explained above. If we consider the symbol $(,)$ to be commutative (C), the terms `siblings(john,sam)` and `siblings(sam,tom)` can be generalized *modulo commutativity* as `siblings(X,sam)`; if the symbol $(,)$ is associative[2] (A), the terms `siblings(tom,sam,john)` and `siblings(john,sam,mary,peter)` can be generalized as `siblings(X,sam,Y)`; if the symbol $(,)$ is both associative and commutative (AC), then the lgg is `siblings(john,sam,Y)`; and if we additionally endow the symbol $(,)$ with an identity element (ACU), so that two structures can be paired at any size by just introducing as many unity elements as needed into the structures, then the corresponding lgg would be `siblings(john,sam,X,Y)`.

The second problem with ordinary generalization is that it does not cope with types and subtypes, which can further increase the number of solutions. For instance, assume that the constants `john`, `sam`, `peter`, and `tom` belong to type Male and that `mary`, `joan` belong to type Female. Also assume that Male, Female are subtypes of the type Person that

[2]When we work *modulo associativity*, the term `f(a,b,c)` is a shorthand for `f(f(a,b),c)`.

is used for the elements of the relation `siblings`. Then, for the case where $(,)$ is AC, the terms `siblings(john,sam,mary,peter,chris)` and `siblings(tom,sam,john,joan)` have two least general generalizers instead of one: `siblings(john,sam,X:Male,Y:People)` and `siblings(john,sam,X:Female,Y:People)`.

As has been shown, order-sorted modular ACU least general generalization provides a solution for these two classical limitations of generalization while remaining decidable[3]. In consequence, the extension to generalization presented in this thesis offers a significant increase in the ability to reason about common data structures in a natural way, thus opening up new applications for typed equational reasoning systems and typed rule-based languages such as ASF+SDF, Elan, OBJ, Cafe-OBJ, and Maude, where some function symbols may be declared to obey given algebraic laws (the so-called *equational attributes*) of associativity and/or commutativity and/or identity.

## 5.5 Experiments

We have tested our tool with several generalization problems that can be found at the ACUOS Web site and within the distributed package. This also includes the generalization of XML schemata, that can be naturally interpreted as terms built using associative-commutative operators with unbounded arity, which we modeled by using ACU constructor symbols in Maude as well.

With regard to the time required to achieve the lggs, and considering the combinatorial complexity of the ACU generalization problem, our implementation is reasonably time efficient. For example, running the tool for the `siblings` example described in the previous section –with ACU rewrites– took less than $10^{-2}$ second (480.000 rewrites per second on standard hardware, 2.26GHz Intel Core 2 Duo with 8Gb of RAM memory). The elapsed times for the Rutherford example described in Figure A.1 in Appendix A are obviously higher but still cogent, due to the encoding of higher-order generalization as first order generalization that we used.

[3] As demonstrated by Theorem 4.1.1

In order to facilitate the understanding, a typical generalization session is provided in Appendix A.

# 6

# Conclusions and future work

Generalization is a formal reasoning component of many symbolic frameworks. Order-sorted modular ACU generalization extends ordinary generalization with subtypes and the capability to endow each function symbol with any combination of associativity, commutativity, and identity axioms, making it possible to reason about records, lists, sets and multisets of data elements, atoms or rules.

In this thesis, we have presented an order-sorted, modular equational generalization algorithm that computes a minimal and complete set of least general generalizations for two terms modulo any combination of associativity, commutativity and identity axioms for the binary symbols in the theory. Our algorithm is directly applicable to any 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 $E$-generalizations, 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., (Alpuente et al., 1998a,b, 1999; Albert et al., 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, 1980b) and its ACL2

successor (Kaufmann et al., 2000).

# Appendices

# A

# A Generalization Session with ACUOS

In this appendix, we reproduce a typical session with the ACUOS system, using its Web front-end to analyze and extract structural commonalities between two patterns in different domains of interest.

**Solar System**
mass(sun) ;
mass(planet) ;
distant(sun,planet) ;
mass(x) ∧ mass(y) ⇒ gravity(x,y) ;
gravity(x,y) ⇒ attraction(x,y)

**Rutherford Atom**
charge(electron) ;
charge(nucleus) ;
distant(electron,nucleus) ;
charge(y) ∧ charge(x) ⇒ coulomb(x,y) ;
coulomb(x,y) ⇒ attraction(x,y)

**Background Knowledge**
distant is commutative
∧ is associative-commutative
_;_ is associative-commutative with unity symbol ∅
`sun`, `planet`, `nucleus`, and `electron`: Elem,
`distant`, `gravity`, `attraction`, `charge`, `coulomb` :
Elem × Elem → Atom.
Atom < Conj .
Rule < Model .
∧: Conj × Conj → Conj.
⇒: Conj × Atom → Rule.
_;_: Model × Rule → Model.

**Generalization**
P(X) ;
P(Y) ;
distant(X,Y) ;
P(x) ∧ P(y) ⇒ Q(x,y) ;
Q(x,y) ⇒ attraction(x,y)

Figure A.1: A formalization of the Rutherford analogy (fragment)

As a motivating example, consider the Rutherford analogy of Figure A.1, where we provide a (term) representation for the solar system and the Rutherford model for the atom. Assuming the equational properties for the symbols given in Figure A.1, by using ACUOS we can automatically deduce a generalization of both models also shown in Figure A.1, where variables introduced by generalization are written in up-

percase and are different from specification variables written in lowercase.

The detailed Maude representation for the Rutherford's analogy problem sketched in Figure A.1 is as follows. First, we define a module `RUTHERFORD-SYNTAX` that specifies the sorts and equational attributes for all symbols in the domain. Maude syntax is almost self-explanatory, using explicit keywords such as `fmod`, `sort`, and `op` to respectively introduce a module, sort, and operator. The keywords `assoc`, `comm`, and `id` indicate associativity, commutativity, and unity of an operator. Note that we simply encode both models of the Rutherford's analogy as Maude terms using the meta-representation described in the module `RUTHERFORD-SYNTAX`, which avoids the need for higher-order generalization.

```
fmod RUTHERFORD-SYNTAX is
  sort HOTerm . subsort Operator < HoTerm .
  op _[_] : Operator HOTermList -> HOTerm .

  sort HOTermList . subsort HOTerm < HOTermList .
  op _,_ : HOTermList HOTermList -> HOTermList [assoc] .

  sort Operator .
  ops mass sun planet distant gravity attraction
      coulomb electron nucleus x y : -> Operator .

  sort Conj . subsort HOTerm < Conj .
  op _/\_ : Conj Conj -> Conj [assoc comm] .

  sort Rule . subsort HOTerm < Rule .
  op _=>_ : Conj HOTerm -> Rule .

  sort Model . subsort HOTerm < Model .
  op _;_ : Model Model -> Model [assoc comm id: empty] .
endfm

fmod RUTHERFORD-DOMAINS is
  --- Omitted for brevity
  --- Contains definitions for:
  ---   solar-system-domain
  ---   atom-domain
endfm
```

The ACUOS on-line Web session starts by loading the Maude module that contains a (meta-level representation of the) formal specification of the considered domains. The specification can be directly pasted into the input form, loaded from a file in the user's machine, or recovered from the gallery of examples provided by the Web interface. For this example we have chosen the latter option. In Figure A.2 below, the ACUOS system is fed with this simple specification of the Rutherford's analogy.

The next action must be to enter the terms to be generalized. Then, the model analogy is automatically computed by simply generalizing these terms modulo the sort structure and the equational properties associated to the symbols given in the module `RUTHERFORD-SYNTAX`. For simplicity, we have defined two aliases for these models in the specification itself: `solar-system-domain` and `atom-domain`. The process ends by clicking in the "Generalize!" button, which runs the ACUOS tool on the provided input.

The generalization call that it is internally made is as follows:

```
metaGeneralize(
  upModule('RUTHERFORD-SYNTAX, true),
  upTerm(mass [sun] ;
         mass [planet] ;
         distant [sun, planet] ;
         mass [x] /\ mass [y] => gravity [x,y] ;
         gravity [x,y] => attraction [x,y]),
  upTerm(charge [electron] ;
         charge [nucleus] ;
         distant [nucleus, electron] ;
         charge [x] /\ charge [y] => coulomb [x,y] ;
         coulomb [x,y] => attraction [x,y])
) .
```

As a result, our tool produces a new term which generalizes the two input terms, i.e., common patterns are preserved while discrepancies are removed, thereby obtaining an analogy between the two domains of interest. The resulting least general generalizer of the two terms is shown with sugared syntax in Figure A.1.

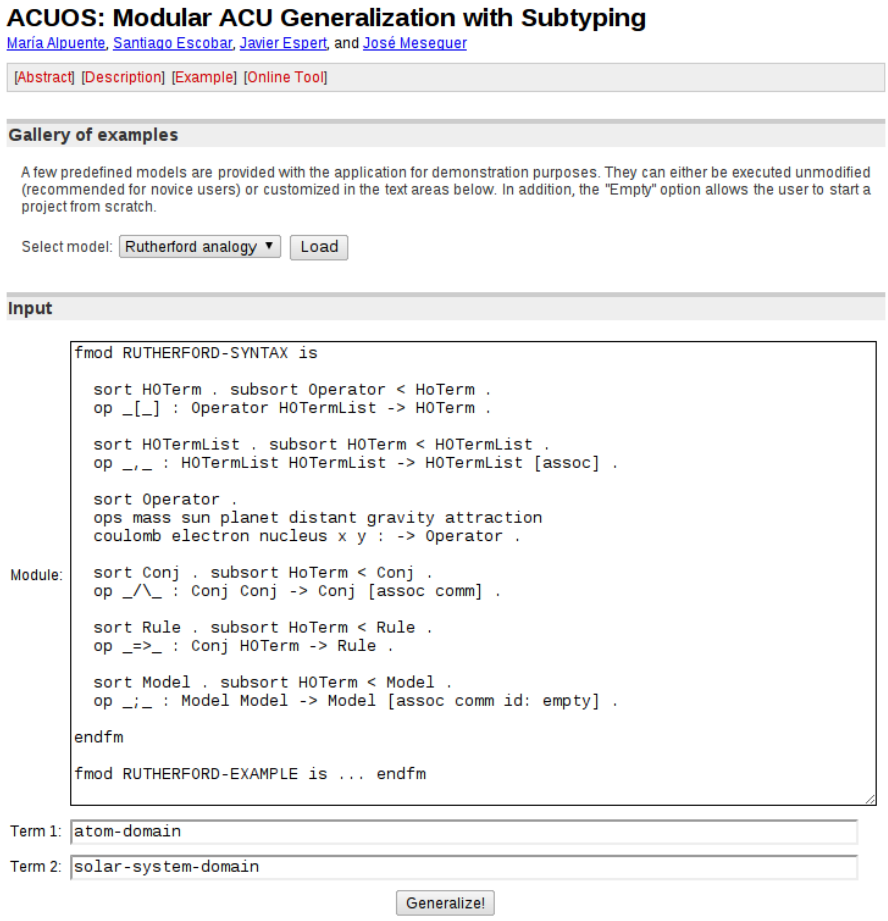


Figure A.2: The input form of the ACUOS generalization Web tool

# Bibliography


Aït-Kaci, H. (1983). Outline of a calculus of type subsumption. Technical report, University of Pennsylvania.

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

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

Albert, E., Alpuente, M., Hanus, M., and Vidal, G. (1999). A partial evaluation framework for curry programs. In Ganzinger, H., McAllester, D. A., and Voronkov, A., editors, *Logic Programming and Automated Reasoning, 6th International Conference, LPAR'99, Tbilisi, Georgia, September 6-10, 1999, Proceedings*, volume 1705 of *Lecture Notes in Computer Science*, pages 376–395. Springer.

Alpuente, M., Escobar, S., Espert, and J., Meseguer, J. (2012). A modular order-sorted equational generalization algorithm.

Alpuente, M., Escobar, S., Meseguer, J., and 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*, volume 5438 of *Lecture Notes in Computer Science*, pages 24–39. Springer.

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

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

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

Alpuente, M., Hanus, M., Lucas, S., and 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*, pages 273–283.

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

Baader, F. and Snyder, W. (1999). Unification theory. In *Handbook of Automated Reasoning*. Elsevier.

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

Bergstra, J., Heering, J., and Klint, P. (1989). *Algebraic Specification*. ACM Press.

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

Boyer, R. and Moore, J. (1980a). *A Computational Logic*. Academic Press.

Boyer, R. and Moore, J. (1980b). *A Computational Logic*. In (Boyer and Moore, 1980a).

Bulychev, P. E., Kostylev, E. V., and 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., and Talcott, C. (2007). *All About Maude - A High-Performance Logical Framework*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Clavel, M. and Palomino, M. (2005). The ITP tool's manual. Universidad Complutense, Madrid, http://maude.sip.ucm.es/itp/.

Diaconescu, R. and Futatsugi, K. (1998). *CafeOBJ Report*, volume 6 of *AMAST Series in Computing*. World Scientific, AMAST Series.

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

Frisch, A. M. and Jr., C. D. P. (1990). Generalization with taxonomic information. In *AAAI*, pages 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*, pages 88–98, New York, NY, USA. ACM.

Goguen, J. and 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., and Jouannaud, J.-P. (2000). Introducing OBJ. In *Software Engineering with OBJ: Algebraic Specification in Action*, pages 3–167. Kluwer.

Huet, G. (1976). *Resolution d'Equations dans des Langages d'Order 1, 2,...,ω*. PhD thesis, Univ. Paris VII.

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

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

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

Kitzelmann, E. and 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., and Villaret, M. (2011). Anti-unification for unranked terms and hedges. In Schmidt-Schauß, M., editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia*, volume 10 of *LIPIcs*, pages 219–234. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

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

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

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

Meseguer, J. (1992). Conditioned rewriting logic as a united model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155.

Meseguer, J. (1997). Membership algebra as a logical framework for equational specification. In *Proc. WADT'97*, pages 18–61.

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

Meseguer, J., Goguen, J., and 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. Technical Report DART/04/04, Norwegian Computing Center. Available at `http://publications.nr.no/nr-notat-dart-04-04.pdf`.

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*, pages 74–85. IEEE Computer Society.

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

Plotkin, G. (1970). A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press.

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*, volume 5, pages 203–215. Edinburgh University Press.

Pottier, L. (1989). Generalisation de termes en theorie equationelle: Cas associatif-commutatif. Technical Report 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*, pages 538–552. Springer-Verlag. LNCS, Volume 230.

Siekmann, J. (1989). Unification Theory. *Journal of Symbolic Computation*, 7:207–274.

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

Smolka, G., Nutt, W., Goguen, J., and Meseguer, J. (1989). Order-sorted equational computation. In Nivat, M. and Aït-Kaci, H., editors, *Resolution of Equations in Algebraic Structures*, volume 2, pages 297–367. Academic Press.

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