Replace this file with prentcsmacro.sty for your meeting,
or with entcsmacro.sty for your meeting. Both can be
found at the ENTCS Macro Home Page.

Order-Sorted Generalization [5]

María Alpuente [a,1] Santiago Escobar [a,2] José Meseguer [b,3]
Pedro Ojeda [a,4]

[a] Dpto. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Spain

[b] Department of Computer Science, University of Illinois at Urbana-Champaign, USA.

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 work, we extend the known untyped generalization algorithm to an order-sorted typed setting with sorts, subsorts, and subtype polymorphism. Unlike the untyped case, there is in general no single lgg. 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 algorithm is expressed by means of an inference system for which we give a proof of correctness. This opens up new applications to partial evaluation, program synthesis, and theorem proving for typed reasoning systems and typed rule-based languages such as ASF+SDF, Elan, OBJ, Cafe-OBJ, and Maude.

Keywords: least general generalization, partial evaluation, order–sorted reasoning

1 Introduction

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 least general generalization (lgg), that is, a generalization that is a substitution instance of any other one. Generalization is a formal reasoning component of many program analysis and transformation methods, including theorem provers, and program analysis and transformation tools (see, e.g., [12,22,8,24]).

Although generalization goes back to work of Plotkin [25], Reynolds [27], and Huet [14] and has been studied in detail by other authors (see for example the survey [17]), to the best of our knowledge, all generalization algorithms, with the exception

[1] Email: alpuente@dsic.upv.es
[2] Email: sescobar@dsic.upv.es
[3] Email: meseguer@cs.uiuc.edu
[4] Email: pojeda@dsic.upv.es
[5] This work has been partially supported by the EU (FEDER) and the Spanish MEC TIN2007-68093-C02-02 project, UPV PAID-06-07 project, and Generalitat Valenciana GV06/285 and BFPI/2007/076 grants.

©2008 Published by Elsevier Science B. V.

of the works of Pfenning on generalization in the higher-order setting of the calculus of constructions [24], assume an untyped setting. However, many applications, for example to partial evaluation, theorem proving, and program learning, for typed rule-based languages such as ASF+SDF [6], Elan [7], OBJ [13], CafeOBJ [11], and Maude [9], 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.

In a way similar to the dual case of order-sorted unification, a case which, in contrast, has indeed been studied in detail (see, e.g. [28,21,30]), the extension of the generalization algorithm to the order-sorted setting is nontrivial. In particular, the existence and uniqueness of generalizations is typically lost. That is, first of all there is no lgg at all if two terms are unrelated in the sort ordering; and if they are related (their sorts are both in the same connected component of the partial order of sorts), then there is in general no single lgg associated to a pair of terms. Instead, there is 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, e.g., [28,21,30], and for equational unification (see, e.g., [5,29]). Note that this situation is quite different from the higher-order typed generalization algorithm of Pfenning [24], where for any two higher-order patterns, either there is no lgg (because the types are incomparable), or there is a unique lgg. A related definition of generalization is given in [1] for feature terms, an extended notion of terms that is also based on ordered sorts.

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., [18,15]), we specify the generalization process by means of an inference system rather than by an imperative-style algorithm. Even for the known untyped generalization case, which we present as a special case to motivate its order-sorted extension, this has several expository and conceptual advantages, and we give an inference system that to the best of our knowledge is new. After this, we show how our unsorted calculus naturally extends to the new order-sorted generalization algorithm. We illustrate the use of the inference rules with several examples. Finally, we give a proof of correctness of our inference system.

As already mentioned, this opens up new applications to partial evaluation, program synthesis, and inductive theorem proving for first-order typed rule-based languages such as ASF+SDF, Elan, OBJ, CafeOBJ, and Maude, and to theorem proving tools, program learning tools, and partial evaluators for such languages. In our own work, we plan to use the above order-sorted 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 based PE techniques developed for the untyped setting in, e.g., [3,4]. We are also considering adding this generalization mechanism to an inductive theorem prover such a Maude's ITP [10] to support automatic conjecture of lemmata. This will provide a typed analogue of similar automatic lemma conjecture mechanisms in untyped first-order inductive theorem provers such as Nqthm [8] and its ACL2 successor [16].

Related work

Plotkin [25] and Reynolds [27] gave an imperative–style algorithm for generalization, which are both essentially the same. Huet's generalization algorithm [14], formulated as a pair of recursive equations, cannot be understood as an automated calculus. A deterministic reconstruction of Huet's algorithm is given in [23] which does not consider types either. An operational definition of the least general generalization of clauses based on (order–sorted) feature terms is given in [1]. Finally, the algorithm for generalization in the calculus of constructions of [24] cannot be used for order-sorted theories.

2 Preliminaries

We follow the classical notation and terminology from [31] for term rewriting and from [19,20] for rewriting logic and order-sorted notions. We assume an *order-sorted signature* Σ with a finite poset of sorts $(\mathsf{S}, \leq)$ and a finite number of function symbols. We furthermore assume 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 component of s; and (ii) for each operator declaration $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$ in Σ, there is also a declaration $f : [\mathsf{s}_1] \times \ldots \times [\mathsf{s}_n] \to [\mathsf{s}]$. Throughout this paper, we assume that Σ has no *ad-hoc operator overloading*, i.e., any two operator declarations for the same symbol f with equal number of arguments, $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$ and $f : \mathsf{s}'_1 \times \ldots \times \mathsf{s}'_n \to \mathsf{s}'$, must necessarily have $[\mathsf{s}_1] = [\mathsf{s}'_1], \ldots, [\mathsf{s}_n] = [\mathsf{s}'_n], [\mathsf{s}] = [\mathsf{s}']$. We assume an 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. A *fresh* variable is a variable that appears nowhere else. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ is the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort s. We write $\mathcal{T}(\Sigma, \mathcal{X})$ and $\mathcal{T}(\Sigma)$ for the corresponding term algebras. We assume that $\mathcal{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ for every sort s.

For a term t, we write $\mathit{Var}(t)$ for the set of all variables in t. Term positions are represented as strings of natural numbers and are endowed with the prefix ordering $\leq$ on strings. The set of positions of a term t is written $\mathit{Pos}(t)$, and the set of non-variable positions $\mathit{Pos}_\Sigma(t)$. The root position of a term is Λ. The subterm of t at position p is $t|_p$ and $t[u]_p$ is the term t where $t|_p$ is replaced by u. By $\mathit{root}(t)$ we denote the symbol occurring at the root position of t.

A *substitution* σ is a sorted mapping from a finite subset of $\mathcal{X}$, written $\mathit{Dom}(\sigma)$, to $\mathcal{T}(\Sigma, \mathcal{X})$. The set of variables introduced by σ is $\mathit{Ran}(\sigma)$. The identity substitution is id. Substitutions are homomorphically extended to $\mathcal{T}(\Sigma, \mathcal{X})$. The application of a substitution σ to a term t is denoted by $t\sigma$. The restriction of σ to a set of variables V is $\sigma|_V$. Composition of two substitutions is denoted by juxtaposition, i.e.,

$\sigma\sigma'$. We call a substitution σ a *renaming* if there is another substitution σ^{-1} such that $\sigma\sigma^{-1}|_{Dom(\sigma)} = id$. Substitutions are sort–preserving, i.e., for any substitution σ, if $x \in \mathcal{X}_{\mathsf{s}}$, then $x\sigma \in \mathcal{T}_{\Sigma}(\mathcal{X})_{\mathsf{s}}$.

We write the sort associated to a variable explicitly with a colon and the sort, i.e. $x{:}\mathsf{Nat}$. We assume *pre-regularity* of the signature Σ, ensuring that every term t has a unique least sort, 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 s and s' in the same connected component, the set of least upper bound sorts of s and s', although not necessarily a singleton set, always exists and is denoted by $LUBS(\mathsf{s}, \mathsf{s}')$.

3 Untyped Least General Generalization

We revisit untyped generalization, going back to Plotkin [25], Reynolds [27], and Huet [14], giving a new inference system that will be useful in our subsequent extension of this algorithm to the order–sorted setting given in Section 4. Throughout this section, we assume terms $t \in \mathcal{T}_{\Sigma}(\Sigma, \mathcal{X})$ for Σ an unsorted signature (i.e., there is only one sort).

Let $\leq$ be the standard instantiation quasi–ordering on terms given by the relation of being "more general", i.e. t is more general than s (i.e. s is an instance of t), written $t \leq s$, iff there exists θ such that $t\theta = s$. The most general unifier of a (unifiable) set M is the least upper bound (most general instance) of M under $\leq$. The less general generalization corresponds to the greatest lower bound. Given a non–empty set M of terms, the term w is a generalization of M if, for all $s \in M$, $w \leq s$. A term w is the least general generalization of M if w is a generalization of M and, for each other generalization u of M, $u \leq w$.

The non-deterministic generalization algorithm λ of Huet [14] (also treated in detail in [17]) is as follows. Let Φ be any bijection between $\mathcal{T}(\Sigma, \mathcal{X}) \times \mathcal{T}(\Sigma, \mathcal{X})$ and a set of variables V. The recursive function λ on $\mathcal{T}(\Sigma, \mathcal{X}) \times \mathcal{T}(\Sigma, \mathcal{X})$ that computes the lgg of two terms is given by:

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

Central to this algorithm is the global function Φ that is used to guarantee that the same disagreements are replaced by the same variable in both terms.

In the following, we provide a *novel* set of inference rules for computing the least generalization (lgg) of two terms, avoiding implicit assumptions by using a store of already solved generalization sub-problems. This algorithm can also be used (thanks to associativity and commutativity of lgg) to compute the lgg of an arbitrary set of terms by successively computing the lgg of two elements of the set in the obvious way.

In our reformulation, 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 (most general) generalization of s and t. By means of this representation, any generalization w of s and t is given by a substitution θ such that $x\theta = w$.

We compute the least general generalization of s and t by means of a transi-

tion system $(\mathit{Conf}, \rightarrow)$ [26] where *Conf* is a set of *configurations* and the transition relation $\rightarrow$ is given by a set of inference rules. Besides the *constraint component*, i.e., a set of constraints of the form $t_i \overset{x_i}{\triangleq} t_{i'}$, and the *substitution component*, i.e., the partial substitution computed so far, configurations also include an extra component, called the *store*. This store[6] plays the role of the function Φ of Huet's generalization algorithm, with the difference that our stores are local to the system configurations, whereas Φ can instead be understood as a global repository. We note that the non–globality of the store will be the key for computing a minimal and complete set of solutions for the order–sorted case.

Definition 3.1 A configuration, written as $\langle C \mid S \mid \theta \rangle$, consists of three components:

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

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

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

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

Note that the inference rules of Figure 1 are non–deterministic (i.e., they depend on the chosen constraint of the set C). However, they are confluent up to variable renaming (i.e., the chosen transition is irrelevant for computation of terminal configurations). This justifies that the least general generalization of two terms is unique up to variable renaming [17].

[6] Our notion of store appears to be comparable to the history set *Hist* of [1], though we came up with the idea of store independently.

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

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

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

Figure 1. Rules for least general generalization

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

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

Example 3.2 Let $t = f(g(a), g(y), a)$ and $s = f(g(b), g(y), b)$ be two terms. We apply the inference rules of Figure 1 and the substitution obtained by the lgg algorithm is $\theta = \{x \mapsto f(g(x_4), g(y), x_4), x_1 \mapsto g(x_4), x_2 \mapsto g(y), x_5 \mapsto y, x_3 \mapsto x_4\}$, where the lgg is $x\theta = f(g(x_4), g(y), x_4)$. Note that variable x_4 is repeated, to ensure the least general generalization. The execution trace is showed in Figure 2.

Termination and confluence (up to variable renaming) of the transition system $(Conf, \rightarrow)$ are straightforward.

Theorem 3.3 *Every derivation stemming from an initial configuration* $\langle t \overset{x}{\triangleq} s \mid \emptyset \mid id\rangle$ *terminates.*

Proof Let $|u|$ be the number of symbol occurrences in a term u. Since the minimum of $|t|$ and $|s|$ is an upper bound to the number of times that the inference rules can

be applied, then the derivation terminates. $\square$

Before proving soundness and completeness of the above inference rules, we need the auxiliary concepts of a conflict position and of conflict pairs, and three auxiliary lemmas. 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 $root(t|_p) \not\equiv root(t'|_p)$ and for all $q < p$, $root(t|_q) \equiv root(t'|_q)$, and the pair $(t|_p, t'|_p)$ is then called a *conflict pair of t and t'*. Also, note that given a constraint $t \overset{x}{\triangleq} t'$, x is always a (most general) generalization of t and t'.

Lemma 3.4 *Given terms t and t' and a fresh variable x such that $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle \emptyset \mid S \mid \theta\rangle$, a constraint $u \overset{y}{\triangleq} v$ is in S iff there exists a conflict position p of t and t' such that $t|_p = u$ and $t'|_p = v$.*

Lemma 3.5 *Given terms t and t' and a fresh variable x such that $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle C \mid S \mid \theta\rangle$, then $x\theta$ is a generalization of t and t'.*

Lemma 3.6 *Given terms t and t' and a fresh variable x such that $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle \emptyset \mid S \mid \theta\rangle$, then $\{y \in \mathcal{X} \mid \exists u \overset{y}{\triangleq} v \in S\} \subseteq Ran(\theta)$, and $Ran(\theta) = Var(x\theta)$.*

Soundness and completeness is proved as follows.

Theorem 3.7 *Given terms t and t' and a fresh variable x, u is the lgg of t and t' if and only if $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle \emptyset \mid S \mid \theta\rangle$ and there is a renaming ρ s.t. $u\rho = x\theta$.*

Proof We rely on the already known existence and uniqueness of the lgg of t and t' and reason by contradiction. By Lemma 3.5, $x\theta$ is a generalization of t and t'. If $x\theta$ is not the lgg of t and t', then there is a term u which is the lgg of t and t' and a substitution ρ such that is not a variable renaming and $x\theta\rho = u$. Since, by Lemma 3.6, $Ran(\theta) = Var(x\theta)$, we can always choose ρ with $Dom(\rho) = Var(x\theta)$. If ρ is not a variable renaming, either:

(i) there are variables $y, y' \in Var(x\theta)$ and a variable z such that $y\rho = y'\rho = z$, or

(ii) there is a variable $y \in Var(x\theta)$ and a non-variable term v such that $y\rho = v$.

In case (i), 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 3.4 and 3.6. In case (ii), 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'. But this is impossible by Lemmas 3.4 and 3.6. $\square$

4 Order–sorted Least General Generalizations

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

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 $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle$ where $[\mathsf{s}] = [LS(t)] = [LS(t')]$, configurations are transformed until a terminal configuration $\langle \emptyset \mid S \mid \theta\rangle$ is reached. In the order–sorted setting the lgg

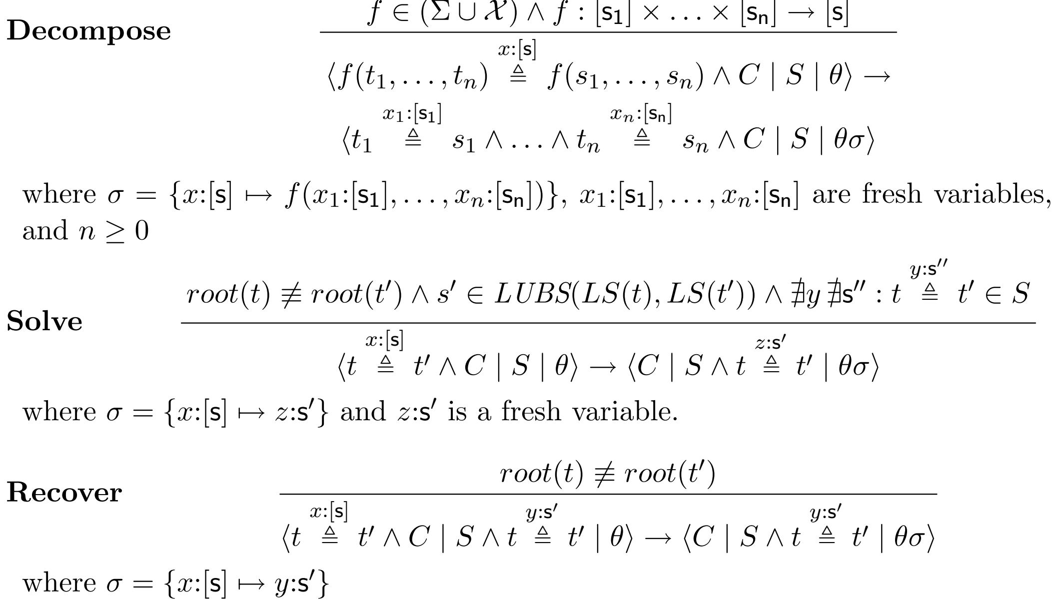

Figure 3. Rules for order–sorted least general generalizations.

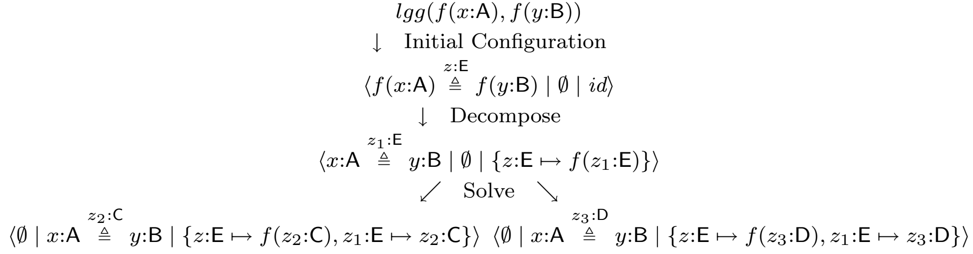

Figure 4. Computation trace for order–sorted generalization of terms $f(x)$ and $f(y)$

in general, is not unique. Each terminal configuration $\langle \emptyset \mid S \mid \theta \rangle$ provides an lgg of t and t' given by $(x{:}[\mathsf{s}])\theta$.

The transition relation $\rightarrow$ is given by the smallest relation satisfying the rules in Figure 3. 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** is similar to the corresponding rule of Figure 1.
- The rule **Solve** checks that a constraint $t \overset{y}{\triangleq} t' \in C$, with $root(s) \not\equiv root(t)$, is not already solved. Then the solved constraint $t \overset{y}{\triangleq} t'$ is added to the store S, and the substitution $\{x \mapsto z\}$ is composed 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. This extra non–determinism causes our rules to be non–confluent in general.

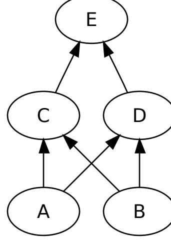

Figure 5. Sort hierarchy

Example 4.1 Let $t = f(x{:}\mathsf{A})$ and $s = f(y{:}\mathsf{B})$ be two terms where x and y are variables of sorts A and B respectively, and the sort hierarchy is shown in Figure 5. 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 3 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 $\theta_2 = \{z{:}\mathsf{E} \mapsto f(z_3{:}\mathsf{D}), z_1{:}\mathsf{E} \mapsto z_3{:}\mathsf{D}\}$, where the lgg is either $(z{:}\mathsf{E})\theta_1 = f(z_2{:}\mathsf{C})$ or $(z{:}\mathsf{E})\theta_2 = f(z_3{:}\mathsf{D})$. Note that θ_1 and θ_2 are incomparable, so that we have two posible lggs. The computation of both solutions is shown in Figure 4.

Before proving the correctness of the above inference system, we give an abstract characterization of the set of lggs of two terms t and t' such that $[LS(t)] = [LS(t')]$. To simplify our notation, in what follows, we write $t[s]_{p_1,\ldots,p_n}$ instead of $((t[s]_{p_1})\cdots)[s]_{p_n}$.

Definition 4.2 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, and let $[\mathsf{s_i}] = [LS(u_i)] = [LS(v_i)]$. We define the term $lgg^\bullet(t,t') = ((t[x_1{:}[\mathsf{s_1}]]_{p^1_1,\ldots,p^1_{n_1}})\cdots)[x_k{:}[\mathsf{s_k}]]_{p^k_1,\ldots,p^k_{n_k}}$, where $x_1{:}[\mathsf{s_1}], \ldots, x_k{:}[\mathsf{s_k}]$ are fresh variables. Furthermore, we define

$$Spec(t,t') = \{\rho \mid Dom(\rho) = \{x_1{:}[\mathsf{s_1}], \ldots, x_k{:}[\mathsf{s_k}]\} \wedge \forall 1 \leq i \leq k, \rho(x_i{:}[\mathsf{s_i}]) = x_i{:}\mathsf{s'_i} \wedge \mathsf{s'_i} \in LUBS(LS(u_i), LS(v_i))\}$$

where all the $x_i{:}\mathsf{s'_i}$ are fresh variables, and, finally, $lgg(t,t') = \{lgg^\bullet(t,t')\rho \mid \rho \in Spec(t,t')\}$.

Lemma 4.3 *Given terms t and t' such that $[LS(t)] = [LS(t')]$, $lgg^\bullet(t,t')$ is a generalization of t and t' and $lgg(t,t')$ provides a complete minimal set of lggs.*

We provide some auxiliary notions and lemmas.

Lemma 4.4 *Given terms t and t' such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, and a fresh variable $x{:}[\mathsf{s}]$ such that $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle \emptyset \mid S \mid \theta\rangle$, a constraint $u \overset{z}{\triangleq} v$ is in S iff there exists a conflict position p of t and t' such that $t|_p = u$ and $t'|_p = v$, and there exist a variable name y and a sort $s \in LUBS(LS(u), LS(v))$ such that $z = y{:}s$.*

A substitution δ is called *downgrading* if each binding is of the form $x{:}\mathsf{s} \mapsto x'{:}\mathsf{s}'$, where x and x' are variables and $\mathsf{s}' \leq \mathsf{s}$.

Lemma 4.5 *Given terms t and t' such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, and let $lgg^\bullet(t,t')$. Then, for all S and θ such that $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle \emptyset \mid S \mid \theta\rangle$, there exists a downgrading substitution δ such that $lgg^\bullet(t,t')\delta = (x{:}[\mathsf{s}])\theta$.*

Theorem 4.6 *Given terms t and t' such that $[\mathsf{s}] = [LS(t)] = [LS(t')]$, and a fresh variable $x{:}[\mathsf{s}]$, $u \in lgg(t,t')$ is a lgg of t and t' if and only if $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \to^* \langle \emptyset \mid S \mid \theta\rangle$ for some S and θ and there is a renaming ρ s.t. $u\rho = (x{:}[\mathsf{s}])\theta$.*

Proof We reason by contradiction. Both cases *if* and *only if* are similar and we provide only the proof for the *if* case.

Let us assume some S and θ such that there are no $u \in lgg(t,t')$ and renaming ρ s.t. $u\rho = (x{:}[\mathsf{s}])\theta$. For all $u \in lgg(t,t')$, by Definition 4.2, $lgg^\bullet(t,t') \leq u$ with a downgrading substitution. By Lemma 4.5, $lgg^\bullet(t,t') \leq (x{:}[\mathsf{s}])\theta$ with a downgrading substitution. Let δ be the downgrading substitution such that $lgg^\bullet(t,t')\delta = (x{:}[\mathsf{s}])\theta$. For all $u \in lgg(t,t')$, let δ_u be the downgrading substitution such that $lgg^\bullet(t,t')\delta_u = u$. Since there is no renaming between $(x{:}[\mathsf{s}])\theta$ and u and both have a downgrading substitution with $lgg^\bullet(t,t')$, there must be a binding $x{:}\mathsf{s} \mapsto x'{:}\mathsf{s}'$ in δ and a binding $x{:}\mathsf{s} \mapsto x''{:}\mathsf{s}''$ in δ_u s.t. either $\mathsf{s}' < \mathsf{s}''$, $\mathsf{s}'' < \mathsf{s}'$, or $[\mathsf{s}'] \neq [\mathsf{s}'']$. But the three possibilities are impossible by definition, since $\mathsf{s}' < \mathsf{s}''$ contradicts the idea that u is a lgg, $\mathsf{s}'' < \mathsf{s}'$ contradicts Lemma 4.4, and $[\mathsf{s}'] \neq [\mathsf{s}'']$ contradicts both that u is a lgg and Lemma 4.4. □

5 Conclusions and Future Work

We have presented an order–sorted generalization algorithm that computes a minimal and complete set of least general generalizations for two terms. Our algorithm is directly applicable to any many-sorted, and order-sorted declarative language and reasoning system (and also, a fortiori, to untyped languages and systems which have only one sort). However, several such languages – such as ASF+SDF, OBJ, Cafe-OBJ, Elan, and Maude –, as well as various theorem proving systems, also support built–in reasoning modulo frequently occurring equational axioms such as associativivty, commutativity and identity. It would therefore be highly desirable to support order–sorted generalization *modulo* such equational theories. In [2], we have developed a modular algorithm for a parametric family of commonly occurring equational theories, namely, for all theories (Σ, E) such that each binary function symbol $f \in \Sigma$ can have any combination of associativity, commutativity, and identity axioms. It would be very useful to combine the order–sorted and the E–generalization inference systems into a single generalization calculus supporting both types and equational axioms. However, this combination seems to us non–trivial and is left for future work.

In our own work, we plan to extend the current order-sorted, syntactic generalization algorithm presented here to an order–sorted, equational one as a key component of a narrowing-based partial evaluator (PE) for programs in order-sorted rule-based languages such as OBJ, Cafe-OBJ, and Maude. This will make available for such languages useful narrowing–driven PE techniques developed for the syntactic setting in, e.g., [3,4]. We are also considering adding this generalization

mechanism to an inductive theorem prover such a Maude's ITP [10] to support automatic conjecture of lemmas. This will provide a first–order typed analogue of similar automatic lemma conjecture mechanisms in first–order untyped inductive theorem provers such as Nqthm [8] and its ACL2 successor [16].

References

[1] H. Aït-Kaci and Y. Sasaki, An Axiomatic Approach to Feature Term Generalization. Proc. 12th European Conf. on Machine Learning, EMCL '01, pages 1–12. Springer-Verlag, 2001.

[2] M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda. A Modular Equational Generalization Algorithm. Proc. 18th Int'l Symp. on Logic-Based Program Synthesis and Transformation, LOPSTR 2008, pages 151–165, 2008.

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

[4] M. Alpuente, S. Lucas, M. Hanus, and G. Vidal. Specialization of functional logic programs based on needed narrowing. *TPLP*, 5(3):273–303, 2005.

[5] F. Baader and W. Snyder. Unification theory. In *Handbook of Automated Reasoning*. Elsevier, 1999.

[6] J.A. Bergstra, J. Heering, and P. Klint. *Algebraic Specification*. ACM Press, 1989.

[7] P. Borovanský, C. Kirchner, H. Kirchner, and P.-E. Moreau. ELAN from a rewriting logic point of view. *Theor. Comput. Sci.*, 285:155–185, 2002.

[8] R. Boyer and J. Moore. *A Computational Logic*. Academic Press, 1980.

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

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

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

[12] J. P. Gallagher. Tutorial on specialisation of logic programs. In *Proc. 1993 ACM SIGPLAN Symp. on Partial evaluation and Semantics-based Program Manipulation, PEPM '93*, pages 88–98, New York, NY, USA, 1993. ACM.

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

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

[15] J.-P. Jouannaud and C. Kirchner. 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, 1991.

[16] M. Kaufmann, P. Manolios, and J.S Moore. *Computer-Aided Reasoning: An Approach*. Kluwer, 2000.

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

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

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

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

[21] J. Meseguer, J. Goguen, and G. Smolka. Order-sorted unification. *J. Symbolic Computation*, 8:383–413, 1989.

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

[23] B. Østvold. A functional reconstruction of anti-unification. Technical Report DART/04/04, Norwegian Computing Center, 2004. Available at http://publications.nr.no/nr-notat-dart-04-04.pdf.

[24] F. Pfenning. 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, 1991.

[25] G.D. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

[26] G.D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.

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

[28] M. Schmidt-Schauss. Unification in many-sorted equational theories. *Proc. 8th Int'l Conf. on Automated Deduction*. Springer LNCS 230: 538–552, 1986.

[29] J.H. Siekmann. Unification Theory. *Journal of Symbolic Computation*, 7:207–274, 1989.

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

[31] TeReSe, editor. *Term Rewriting Systems*. Cambridge University Press, Cambridge, 2003.