Variant-based Equational Anti-unification*

M. Alpuente[1], D. Ballis[2], S. Escobar[1], and J. Sapiña[1]

[1] VRAIN, Universitat Politècnica de València
Camino de Vera s/n, Apdo 22012, 46071 Valencia, Spain
{alpuente,sescobar,jsapina}@upv.es
[2] DMIF, University of Udine,
Via delle Scienze, 206, 33100, Udine, Italy
demis.ballis@uniud.it

Abstract. The dual of most general equational unifiers is that of least general equational anti-unifiers, i.e., most specific anti-instances modulo equations. This work aims to provide a general mechanism for equational anti-unification that leverages the recent advances in variant-based symbolic computation in Maude. Symbolic computation in Maude equational theories is based on folding variant narrowing (FVN), a narrowing strategy that efficiently computes the equational variants of a term (i.e., the irreducible forms of all of its substitution instances). By relying on FVN, we provide an equational anti-unification algorithm that computes the least general anti-unifiers of a term in any equational theory E where the number of least general E-variants is finite for any given term.

1 Introduction

The concept of anti-unification (also known as generalization) was independently introduced by Plotkin [20] and Reynolds [21]. Anti-unification is relevant in a wide spectrum of automated reasoning techniques and applications where analogical reasoning and inductive inference are needed, such as ontology learning, analogy making, case-based reasoning, web and data mining, theorem proving, machine learning, program derivation, and inductive logic programming, among others [3,18,19]. For instance, the anti-unification algorithm of [14] has been recently used in the generation of fix patterns for automated program repair in Bloomberg's Fixie-learn [13] and Facebook's Getafix [4].

In the purely syntactic and untyped setting of [20,21], the syntactic generalization problem for two or more expressions consists in finding their *least general generalizer (lgg)*, i.e., the least general expression t such that all of the given expressions are instances of t under appropriate substitutions. For instance, consider an alphabet with three constants a, b, and c; three function

*This work was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215, grant PID2021-122830OB-C42 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe", and by Generalitat Valenciana under grant PROMETEO/2019/098.

symbols f, g and h; and variables x, y and z. Also consider the two terms $u = f(b, g(b, b))$ and $v = f(g(z, a), g(g(z, a), b))$. The expression $f(x, g(x, b))$ is the syntactic (and unique) least general generalizer of u and v since both $f(b, g(b, b))$ and $f(g(z, a), g(g(z, a), b))$ are substitution instances of $f(x, g(x, b))$. However, if the function symbol g is given a definition by means of an equational theory E consisting of the equation $g(x, y) = b$, then $g(b, b)$, $g(z, a)$, $g(x, y)$ and b are *"la même chose"* (more formally, they are equal modulo E) and so are $u = f(b, g(b, b))$ and $v = f(g(z, a), g(g(z, a), b))$, hence the least general generalizer of u and v is $f(b, b)$. Note that the syntactic generalizer $f(x, g(x, b))$ of u and v, and its E-equivalent term $f(x, b)$, are more general modulo E than the least general generalizer $f(b, b)$.

Given a set E of equations and two terms u and v to be generalized modulo E, we say that the term t is an E-generalizer of u and v if there are two terms t_1 and t_2, which are substitution instances of t, such that t_1 is equal (modulo E) to u and t_2 is equal (modulo E) to v. An E-generalizer of u and v that is less general than or incomparable to (modulo E) any other E-generalizer of the two terms is called a least general generalizer. The computation of equational least general generalizers is much more involved than syntactic generalization as it may require *guessing* the less general term pattern t and substitutions σ_1 and σ_2 that, when independently applied to t, get two terms $t_1 = t\sigma_1$ and $t_2 = t\sigma_2$ that are equal (modulo E) to two corresponding arguments in u and v. This guessing cannot be done by simple equational reasoning but requires logic-style, symbolic computation. For instance, if the theory E contains the equations $h(x, a) = f(g(x, a), g(g(x, a), b))$ and $h(x, b) = f(b, g(b, b))$ (with g obeying no equation), then $h(x, y)$ is a least general generalizer modulo E of $u = f(b, g(b, b))$ and $v = f(g(z, a), g(g(z, a), b))$. This is because there are two instances of $h(x, y)$ which are equal (modulo E) to u and v, respectively (namely, $\sigma_1 = \{y \mapsto b\}$ and $\sigma_2 = \{x \mapsto z, y \mapsto a\}$).

Similarly to the dual problem of E-unification of two terms, where there may be a set of incomparable, most general E-unifiers, the set of least general anti-unifiers of two terms is not generally singleton. For instance, the syntactic generalizer $f(x, g(x, b))$ of $u = f(b, g(b, b))$ and $v = f(g(z, a), g(g(z, a), b))$ above is still valid with the two equations for h and it is incomparable to $h(x, y)$, so both are least general generalizers. The anti-unification type of a theory can be defined similarly (but dually) to the unification types, i.e., based on the existence and cardinality of a minimal and complete set of least general generalizers [7].

In this work, we address the problem of least general anti-unification in order-sorted equational theories where function symbols are endowed with an equational definition. The intuition behind our least general generalization algorithm is that substitutions σ_1 and σ_2 mentioned above can be computed by *narrowing* most general terms $f(x_1, \cdots, x_n)$ in E, with f being an n-ary function symbol in the theory. Narrowing is a symbolic execution mechanism that generalizes term rewriting by allowing free variables in terms (as in logic programming) and handles them by using unification (instead of pattern matching) to non-deterministically reduce these terms. For instance, given

$E = \{h(x,b) = f(b, g(b,b)),\ h(x,a) = f(g(x,a), g(g(x,a),b))\}$, there are two narrowing steps stemming from the term $h(x,y)$: 1) the term $h(x,y)$ narrows to $f(b, g(b,b))$ with computed narrowing substitution $\sigma_1 = \{y \mapsto b\}$; and 2) the term narrows to $f(g(x,a), g(g(x,a),b))$ with computed narrowing substitution $\sigma_2 = \{y \mapsto a\}$. In the last few years, there has been a resurgence of narrowing in many application areas such as equational unification, state space exploration, protocol analysis, termination analysis, theorem proving, deductive verification, model transformation, testing, constraint solving, and model checking.

Maude [8] is a language and a system that efficiently implements Rewriting Logic (RWL) [15]. Equational theories in Maude may include ordinary equations and algebraic axioms, i.e., distinguished equations expressing algebraic laws such as associativity (A), commutativity (C), and identity (i.e., unity) (U) of function symbols. Algebraic axioms are efficiently handled in Maude in a built-in way. For the sake of simplicity, the equational theories considered in this work do not contain algebraic axioms.

Maude provides quite sophisticated narrowing-based features that rely on built-in generation of the set of *variants* of a term t [10]. Essentially, a *variant* of a term t in the theory E is the canonical (i.e., irreducible in E) form of $t\sigma$ for a given substitution σ. Variants are computed in Maude by using the *folding variant narrowing strategy* [11]. When the theory satisfies the *finite variant property* (i.e., there is a finite number of most general variants for every term in the theory), folding variant narrowing computes a minimal and complete set of most general variants in a finite amount of time. Many theories of interest have the FVP, including theories that give algebraic axiomatizations of cryptographic functions used in communication protocols, where FVP is omni-present.

As far as we know, this is the first general, theory-independent algorithm for computing least general anti-unifiers modulo equational theories in *Plotkin's style*. A theory-agnostic E-generalization algorithm based on regular tree grammars is formalized by Burghardt in [5] that computes a finite representation of E-generalization sets. However, Burghardt's algorithm is restricted to equational theories E that induce regular congruence classes (i.e., the theory E is the deductive closure of finitely many ground equations). We establish that the novel algorithm that we propose in this paper is minimal, correct and complete (i.e., it computes a complete and minimal set of least general generalizers for any anti-unification problem). A prototype implementation in Maude [CDE+07] is currently under development.

In [1,2], we extended the classical untyped anti-unification algorithm of [20] to work: (1) modulo any combination of associativity, commutativity, and identity axioms (including the empty set of such axioms); (2) with typed structures that involve sorts, subsorts, and subtype polymorphism; and (3) under any combination of both, which results in a modular, order-sorted, least general anti-unification algorithm modulo algebraic axioms. The algorithm in [1,2] only applies to modular combinations of A, C, and U equational axioms. It cannot be used to solve anti-unification problems in the general, user-defined equational theories considered in this paper.

After some preliminaries in Section 2, in Section 3 we address the problem of generalizing two (typed) expressions modulo an equational theory, we formulate our least general generalization algorithm, and we illustrate it by means of a representative example. Section 4 proves the formal properties of our algorithm. In Section 5, we discuss further work and we conclude. A simple representative application of equational generalization to a biological domain is described in Appendix A.

2 Preliminaries

We follow the classical notation and terminology from [23] for term rewriting and from [16,12] for order-sorted equational logic.

We assume an *order-sorted signature* $\Sigma = (S, F, \leq)$ that consists of a finite poset of sorts $(\mathsf{S}, \leq)$ and a family F of function symbols of the form $f : \mathsf{s}_1 \times \cdots \times \mathsf{s}_n \to \mathsf{s}$, with $\mathsf{s}_1, \cdots, \mathsf{s}_n, \mathsf{s} \in \mathsf{S}$. Two sorts s and s' belong to the same connected component if either $\mathsf{s} \leq \mathsf{s}'$ or $\mathsf{s}' \leq \mathsf{s}$. We assume a *kind-completed signature* such that: (i) each connected component in the poset ordering has a top sort, and, for each $\mathsf{s} \in \mathsf{S}$, we denote by $[\mathsf{s}]$ the top sort in the connected component of s (*i.e.,* if s and s' are sorts in the same connected component, then $[\mathsf{s}] = [\mathsf{s}']$); and (ii) for each operator declaration $f : \mathsf{s}_1 \times \cdots \times \mathsf{s}_n \to \mathsf{s}$ in Σ, there is also a declaration $f : [\mathsf{s}_1] \times \cdots \times [\mathsf{s}_n] \to [\mathsf{s}]$ in Σ. A given term t in an order-sorted term algebra can have many different sorts. Specifically, if t has sort s, then it also has sort s' for any $\mathsf{s}' \geq \mathsf{s}$; and because a function symbol f can have different sort declarations $f : \mathsf{s}_1 \times \cdots \times \mathsf{s}_n \to \mathsf{s}$, a term $f(t_1, .., t_n)$ can have sorts that are not directly comparable [12].

We assume a fixed S-sorted family $\mathcal{V} = \{\mathcal{V}_\mathsf{s}\}_{\mathsf{s} \in \mathsf{S}}$ of pairwise disjoint variable sets (*i.e.,* $\forall \mathsf{s}, \mathsf{s}' \in \mathsf{S} : \mathcal{V}_\mathsf{s} \cap \mathcal{V}_{\mathsf{s}'} = \emptyset$), with each $\mathcal{V}_\mathsf{s}$ being countably infinite. We write the sort associated to a variable explicitly with a colon and the sort, *i.e.,* x:Nat. A *fresh* variable is a variable that appears nowhere else. The set $\mathcal{T}_\Sigma(\mathcal{V})_\mathsf{s}$ denotes all Σ-terms of sort s defined by $\mathcal{V}_\mathsf{s} \subseteq \mathcal{T}_\Sigma(\mathcal{V})_\mathsf{s}$ and $f(t_1, \cdots, t_n) \in \mathcal{T}_\Sigma(\mathcal{V})_\mathsf{s}$ if $f : \mathsf{s}_1 \times \cdots \times \mathsf{s}_n \to \mathsf{s} \in \Sigma$, $n \geq 0$ and $t_1 \in \mathcal{T}_\Sigma(\mathcal{V})_{\mathsf{s}_1}, \cdots, t_n \in \mathcal{T}_\Sigma(\mathcal{V})_{\mathsf{s}_n}$. Furthermore, if $t \in \mathcal{T}_\Sigma(\mathcal{V})_\mathsf{s}$ and $s \leq s'$, then $t \in \mathcal{T}_\Sigma(\mathcal{V})_{\mathsf{s}'}$. For a term t, we write $Var(t)$ for the set of all variables in t. $\mathcal{T}(\Sigma)_\mathsf{s}$ is the set of ground terms of sort s, *i.e., t* is a Σ-term of sort s and $Var(t) = \emptyset$. We write $\mathcal{T}(\Sigma, \mathcal{V}) = \bigcup_{s \in S} \mathcal{T}_\Sigma(\mathcal{V})_\mathsf{s}$ and $\mathcal{T}(\Sigma) = \bigcup_{s \in S} \mathcal{T}(\Sigma)_\mathsf{s}$ for the corresponding term algebras. We assume that $\mathcal{T}(\Sigma)_\mathsf{s} \neq \emptyset$ for every sort s.

We assume *pre-regularity* of the signature Σ: for each operator declaration $f : \mathsf{s_1} \times \cdots \times \mathsf{s_n} \to \mathsf{s}$, and for the set S_f containing all sorts s' that appear in operator declarations of the form $f : \mathsf{s}'_1, \cdots, \mathsf{s}'_n \to \mathsf{s}'$ in Σ such that $\mathsf{s}_i \leq \mathsf{s}'_i$ for $1 \leq i \leq n$, the set S_f has a least sort. Thanks to pre-regularity of Σ, each Σ-term t has a *unique least sort* that is denoted by $LS(t)$. The top sort in the connected component of $LS(t)$ is denoted by $[LS(t)]$. Since the poset $(\mathsf{S}, \leq)$ is finite and each connected component has a top sort, given any two sorts s and s' in the same connected component, the set of least upper bound sorts of s and s' always exists (although it might not be a singleton set) and is denoted by $LUBS(\mathsf{s}, \mathsf{s}')$.

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

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

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

A Σ-*equation* is an unoriented pair $t \doteq t'$, where t and t' are Σ-terms for which there are sorts s, s' with $t \in \mathcal{T}_\Sigma(\mathcal{V})_\mathsf{s}$, $t' \in \mathcal{T}_\Sigma(\mathcal{V})_{\mathsf{s}'}$, and s, s' are in the same connected component of the poset of sorts $(S, \leq)$. An *equational theory* (Σ, E) is a set E of Σ-equations. An *equational theory* (Σ, E) over a kind-completed, pre-regular, and order-sorted signature $\Sigma = (S, F, \leq)$ is called kind-completed, pre-regular, and order-sorted equational theory. Given an equational theory (Σ, E), order-sorted equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}(\Sigma, \mathcal{V})$, see [12,16].

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

Given a set of equations E, $\overrightarrow{E}$ is a set of rewrite rules that result from orienting the equations of E from left to right. We call $(\Sigma, \overrightarrow{E})$ a *decomposition* of an equational theory (Σ, E) if $\overrightarrow{E}$ is *convergent*, i.e., confluent, terminating, and strictly coherent [17], and sort-decreasing. Under these conditions, the equations

in E can be safely interpreted as simplification rules that can be used to compute a unique *E-canonical form* $t{\downarrow_E}$ for every term $t \in \mathcal{T}(\Sigma, \mathcal{V})$.

Given a decomposition $(\Sigma, \overrightarrow{E})$ of an equational theory and a substitution $\theta = \{x_1 \mapsto t_1, \cdots, x_n \mapsto t_n\}$, we let $\theta{\downarrow_{\overrightarrow{E}}} = \{x_1 \mapsto t_1{\downarrow_{\overrightarrow{E}}}, \cdots, x_n \mapsto t_n{\downarrow_{\overrightarrow{E}}}\}$. We say that (t', θ') is an *E-variant* [9,11] (or just a variant) of term t if for some substitution θ, $t' = (t\theta){\downarrow_{\overrightarrow{E}}}$ and $\theta' = \theta{\downarrow_{\overrightarrow{E}}}$. A *complete set of most general E-variants* [11] (up to renaming) of a term t is a subset, denoted by $[\![t]\!]_E$, of the set of all E-variants of t such that, for each E-variant (t', σ') of t, there is an E-variant $(t'', \sigma'') \in [\![t]\!]_E$ such that $t'' \leq_E t'$ and $\sigma'' \leq_E \sigma'$. A decomposition $(\Sigma, \overrightarrow{E})$ has the *finite variant property* (FVP) [11] (also called a *FVP theory*) iff for each Σ-term t, a complete set $[\![t]\!]_E$ of its most general variants is finite.

Finally, we also consider a natural partition of the rewrite theory signature as $\Sigma = \mathcal{D} \uplus \Omega$, where Ω are the *constructor* symbols, which are used to define (irreducible) data values, and $\mathcal{D} = \Sigma \setminus \Omega$ are the *defined* symbols, which are evaluated away via equational simplification. Terms in $\tau(\Omega, \mathcal{V})$ are called *constructor* terms.

3 Least General Anti-unification modulo Equational Theories via Variant Computation

In the following, we recall the order-sorted syntactic generalization algorithm as formalized in [1,2].

3.1 Syntactic Anti-unification

A term t is a syntactic generalizer of t_1 and t_2 if there are two substitutions σ_1 and σ_2 such that $t\sigma_1 = t_1$ and $t\sigma_2 = t_2$.

We represent a generalization problem between terms t and t' as a *constraint* $t \overset{x}{\triangleq} t'$, where x is a fresh variable that stands for a generalizer of t and t', that becomes more and more instantiated as the computation proceeds until becoming a least general generalizer. Given a constraint $t \overset{x}{\triangleq} t'$, any generalizer w of t and t' is given by a suitable substitution θ such that $x\theta = w$.

A set of constraints is represented by $s_1 \overset{x_1}{\triangleq} t_1 \wedge \cdots \wedge s_n \overset{x_n}{\triangleq} t_n$, or $\emptyset$ for the empty set. Given a constraint $t \overset{x}{\triangleq} t'$, we call x a *generalization variable*. We define the set of generalization variables of a set C of constraints as $GVs(C) = \{y \in \mathcal{V} \mid \exists u \overset{y}{\triangleq} v \in C\}$.

Note that, although it is natural to consider that a constraint $t \overset{x}{\triangleq} t'$ is commutative, the inference rules that are described do not admit that commutativity property for $\triangleq$ since we need to keep track of the origin of new generated generalization subproblems. However, the constructor symbol $\wedge$ that we use to build a set (conjunction) of constraints is *associative* and *commutative* in the inference rules described in this paper. Note that there are no defined symbols in the syntactic case, i.e. $\Sigma{=}\Omega$.

Definition 1. *A configuration* $\langle C \mid S \mid \theta \rangle$ *consists of three components: (i) the* constraint component C*, which represents the* set of unsolved constraints*; (ii) the* store component S*, which records the* set of already solved constraints*, and (iii) the* substitution component θ*, which binds some of the generalization variables previously met during the computation.*

Decompose

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

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

Solve

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

where $\sigma = \{x{:}[\mathsf{s}] \mapsto z{:}\mathsf{s}'\}$, $z{:}\mathsf{s}'$ is a fresh variable, and $\mathsf{s}' \in LUBS(LS(t), LS(t'))$

Recover

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

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

Fig. 1: Basic inference rules for order-sorted least general generalization [1]

In Figure 1, we consider any two terms t and t' in a constraint $t \overset{x}{\triangleq} t'$ having the same top sort; otherwise, they are incomparable and no generalizer 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. The transition relation $\to$ on configurations is given by the smallest relation satisfying all of the rules of Figure 1. Due to order-sortedness, in general there can be more than one least general generalizer of two expressions [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 **Decompose** rule is the syntactic decomposition generating new constraints to be solved.
- The **Solve** rule checks that a constraint $t \overset{x}{\triangleq} t' \in C$, with $root(t) \neq root(t')$, is not already solved. If not already in the store S, then the solved constraint

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

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

$t \overset{x}{\triangleq} t'$ is added to S. Note that the **Solve** rule causes branching due to different choices of s', hereby producing multiple least general generalizers.

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

We illustrate the syntactic generalization calculus by means of the following example, where we disregard of sorts for the sake of simplicity.

Example 1. Consider the terms $t = f(g(a),g(y),a)$ and $t' = f(g(b),g(y),b)$. In order to compute the least general generalizer of t and t', we apply the inference rules of Figure 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 2. Note that variable x_4 is repeated to ensure that the least general generalizer is obtained.

3.2 Anti-unification modulo an Equational Theory

Given an equational theory E, a complete set of least general generalizers modulo E of terms u and v can be computed by extending the syntactic least general generalization calculus of Figure 1 with the new rule of Figure 3. Note that the considered extension turns the equational generalization algorithm into a more non-deterministic calculus by independently applying **Solve** and the new rule **Variant** to the same configuration.

For the sake of optimality, we assume that both u and v are canonical forms with respect to $\overrightarrow{E}$; otherwise, we simplify them to canonical form before the E-lgg computation starts so that we ensure that the computed solutions are canonical representatives w.r.t. E of the set of least general equational generalizers.

$$\textbf{Variant}\quad \frac{\begin{array}{c} f : [\mathsf{s_1}] \times \cdots \times [\mathsf{s_n}] \to [\mathsf{s}] \in \mathcal{D} \wedge \\ (t_1, \sigma_1), (t_2, \sigma_2) \in [\![f(x_1{:}[\mathsf{s_1}], \cdots, x_n{:}[\mathsf{s_n}])]\!]_E \wedge \\ u = t_1\rho_1 \wedge v = t_2\rho_2 \end{array}}{\begin{array}{c} \langle u \overset{x:[\mathsf{s}]}{\triangleq} v \wedge C \mid S \mid \theta \rangle \to \\ \langle w_1{\downarrow_{\overrightarrow{E}}} \overset{x_1:[\mathsf{s_1}]}{\triangleq} w'_1{\downarrow_{\overrightarrow{E}}} \wedge \cdots \wedge w_n{\downarrow_{\overrightarrow{E}}} \overset{x_n:[\mathsf{s_n}]}{\triangleq} w'_n{\downarrow_{\overrightarrow{E}}} \wedge C \mid S \mid \theta\sigma \rangle \end{array}}$$

with $\sigma = \{x{:}[\mathsf{s}] \mapsto f(x_1{:}[\mathsf{s_1}], \cdots, x_n{:}[\mathsf{s_n}])\}$, where $x_1{:}[\mathsf{s_1}], \cdots, x_n{:}[\mathsf{s_n}]$ are fresh variables, $w_i = x_i\sigma_1\rho_1$, $w'_i = x_i\sigma_2\rho_2$, $1 \leq i \leq n$, and $n \geq 0$

Fig. 3: Inference rule for variant-based order-sorted equational least general generalization

The novel rule **Variant**, proceeds as follows. Given the equational theory E, we consider the set of most general variants for any "most general" term $f(x_1{:}[\mathsf{s_1}], \cdots, x_n{:}[\mathsf{s_n}])$, with $f : [\mathsf{s_1}] \times \cdots \times [\mathsf{s_n}] \to [\mathsf{s}]$ being any defined function symbol in the theory signature. Recall that this can be easily achieved in Maude by first deploying the finite computation trees of folding variant narrowing for the considered terms and then gathering all of the variants from the nodes of the tree. Then, given the generalization problem $u \overset{x}{\triangleq} v$, we look for two variants (t_1, σ_1) and (t_2, σ_2) in the tree such that u is an instance of t_1 and v is an instance of t_2, i.e. $t_1\rho_1 = u$ and $t_2\rho_2 = v$, since u and v are E-canonical forms. This means that $f(x_1, \cdots, x_n)$ is a generalizer of both u and v, yet it may be too general.

The main idea of the bottom part of the rule is that a less general generalizer of both u and v can be obtained by recursively computing the generalizers of the combined substitutions, $\sigma_1\rho_1$ and $\sigma_2\rho_2$. That is, for each variable $x' \in Dom(\sigma_1 \cup \sigma_2)$, the generalization problem $x\sigma_1\rho_1{\downarrow_{\overrightarrow{E}}} \overset{x'}{\triangleq} x\sigma_2\rho_2{\downarrow_{\overrightarrow{E}}}$ is recursively solved. More precisely, the newly generated anti-unification problems $w_1{\downarrow_{\overrightarrow{E}}} \overset{x_1:[\mathsf{s_1}]}{\triangleq} w'_1{\downarrow_{\overrightarrow{E}}} \wedge \cdots \wedge w_n{\downarrow_{\overrightarrow{E}}} \overset{x_n:[\mathsf{s_n}]}{\triangleq} w'_n{\downarrow_{\overrightarrow{E}}}$ are previously simplified to canonical form w.r.t. E. This implies that: 1) at any computation step, all of the anti-unification problems in the constraint component are in canonical form w.r.t E; 2) It is unnecessary to modify rules **Solve** and **Recover** to semantically ask the store modulo E-equality when checking whether the anti-unification problem at hand was already solved.

It is worth noting that the syntactic rule **Decompose** could be safely removed from the generalization calculus in exchange of considering, in rule **Variant**, any function symbol f of Σ instead of just the defined symbols of $\mathcal{D}$. This is because: 1) the narrowing tree for a constructor term $c(x_1, \cdots, x_n)$ boils down to the very root term; 2) both, u and v, are c-rooted terms and they are instances of the root term $c(x_1, \cdots, x_n)$; 3) the original anti-unification problem for u and v is then replaced by the anti-unification subproblems for the corresponding arguments of the two terms, thus perfectly mimiking the effect of applying rule **Decompose** in this case.

Finally, a minimization post-processing must be performed in order to filter out all of the candidate generalizers that are not least general according to the relative generality ordering $\leq_E$, thus delivering the set of least general order-sorted anti-unifiers in E of the input terms. This is done by choosing a set of maximal elements of the set of all E-generalizers with regard to the ordering $\leq_E$.

Note that it may be the case that the subsumption relation $t \leq_E t'$ is *undecidable*, so that the above set of least general E-generalizers, although definable at the mathematical level, might not be effectively computable. Nevertheless, when: (i) each E-equivalence class is *finite* and can be effectively generated, and (ii) there is an E-matching algorithm, then we also have an effective algorithm for computing $lgg_E(t, s)$, since the relation $\leq_E$ is precisely the E-matching relation.

3.3 An Equational Anti-unfication Example

In [2], we studied generalization modulo algebraic axioms for the modular combinations of associativity, commutativity and identity axioms. Other theories such as idempotence and identity have been studied in [6,7]. In the following, we show how the generic least general generalization algorithm in this paper can be used to solve least general generalization problems modulo identity without resorting to devoted algorithms such as the ones in [2]. It is worth noting that the equational theory of identity has the FVP [11].

Example 2. Given two binary function symbols f and g such that f has an identity element e (i.e., for all x, $f(x, e) = x$ and $f(e, x) = x$) three constants a, b, and c, and the generalization problem $g(f(a, c), a) \stackrel{?}{\triangleq}_w g(c, b)$, the (different) algorithms of [2] and [6] produce the least general generalizer given by $\{w \mapsto g(f(x, c), f(x, y))\}$, where x and y are new variables. Following the new algorithm of this paper with only the two equations for the identity of f, we compute the desired least general generalization $g(f(w_{11}, c), f(w_{11}, w_{22}))$. A detailed computation trace for this example is shown in Figure 4.

In the following section, we formally establish the formal properties of our equational, order sorted, least general anti-unification algorithm.

4 Correctness and Completeness of the Equational Anti-unification Algorithm

We follow the proof scheme of [1,2] and provide the formal proof of the following auxiliary results, which extend the corresponding lemmas in [1,2] to generalization modulo an equational theory.

Lemma 1. *Given terms t and t' and a fresh variable x, if $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figures 1 and 3, then $x\theta$ is a generalizer of t and t' modulo E;*

Proof. By case analysis of each one of the inference rules. In the decompose rule, $x{:}[\mathsf{s}] \mapsto f(x_1{:}[\mathsf{s_1}], \cdots, x_n{:}[\mathsf{s_n}])$ is clearly a more instantiated generalizer than $x{:}[\mathsf{s}]$. In the solve rule, $x{:}[\mathsf{s}] \mapsto z{:}\mathsf{s}'$ for s' a common sort of t and t' is again a more instantiated generalizer than $x{:}[\mathsf{s}]$. In the recover rule, $x{:}[\mathsf{s}] \mapsto y{:}\mathsf{s}'$ for $y{:}\mathsf{s}'$ the variable of an already existing generalization problem is again a more instantiated generalizer than $x{:}[\mathsf{s}]$. In the variant rule, $x{:}[\mathsf{s}] \mapsto f(x_1{:}[\mathsf{s_1}], \cdots, x_n{:}[\mathsf{s_n}])$ is again a more instantiated generalizer than $x{:}[\mathsf{s}]$. □

Lemma 2. *Given terms t and t' and a fresh variable x, if u is a generalizer of t and t' modulo E, then there is a derivation $\langle t \stackrel{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figures 1 and 3, such that u and $x\theta$ are equivalent modulo renaming and modulo E.*

Proof. By induction on the generalizer u. If u is a variable or a constant, the proof is straightforward. If $u = f(u_1, \cdots, u_k)$, $t = f(t_1, \cdots, t_k)$, and $t' = f(t'_1, \cdots, t'_k)$, then the conclusion follows by the induction hypothesis. In this case, if f is a constructor, then the decompose rule should have been applied. And if f is not a constructor symbol, then the variant rule should have been applied but without computing any variant, just the general term $z = f(x_1, \cdots, x_k)$ since both t and t' are instances of z. If $u = f(u_1, \cdots, u_k)$ and either t or t' are not rooted by f, then $\exists\sigma : u\sigma{\downarrow_{\overrightarrow{E}}} = t$ and $\exists\sigma' : u\sigma'{\downarrow_{\overrightarrow{E}}} = t'$ but, by induction hypothesis, for each $i \in \{1, \cdots, k\}$, u_i is a generalizer of $u_i\sigma{\downarrow_{\overrightarrow{E}}}$ and $u_i\sigma'{\downarrow_{\overrightarrow{E}}}$ such that there are derivations using the inference rules of Figures 1 and 3. Since $w = f(u_1\sigma{\downarrow_{\overrightarrow{E}}}, \cdots, u_k\sigma{\downarrow_{\overrightarrow{E}}})$ and $w' = f(u_1\sigma'{\downarrow_{\overrightarrow{E}}}, \cdots, u_k\sigma'{\downarrow_{\overrightarrow{E}}})$ are instances of a very general term $z = f(x_1, \cdots, x_k)$, there are variants (v_1, θ_1) and (v_2, θ_2) as well as substitutions ρ_1 and ρ_2 such that $w = z\theta_1\rho_1{\downarrow_{\overrightarrow{E}}}$ and $w' = z\theta_2\rho_2{\downarrow_{\overrightarrow{E}}}$. But then, the variant inference rule can be applied and the conclusion follows from the derivations for each pair $u_i\sigma{\downarrow_{\overrightarrow{E}}}$ and $u_i\sigma'{\downarrow_{\overrightarrow{E}}}$. □

By using the above lemmata, correctness and completeness follow.

Theorem 1 (Correctness). *Given a kind-completed, order-sorted equational FVP theory (Σ, E) and a generalization problem $\Gamma = t \stackrel{x:[\mathsf{s}]}{\triangleq} t'$, with $[\mathsf{s}] = [LS(t)] = [LS(t')]$, such that t and t' are Σ-terms, if $\langle t \stackrel{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid$*

$S \mid \theta\rangle$ using the inference rules of Figures 1 and 3, then $(x{:}[\mathsf{s}])\theta$ is a generalizer of t and t' modulo E. By applying the minimization post-processing, only least general generalizers are delivered, which ensures correctness.

Theorem 2 (Completeness). *Given a kind-completed, order-sorted equational FVP theory (Σ, E) and a generalization problem $\Gamma = t \overset{x:[\mathsf{s}]}{\triangleq} t'$, with $[\mathsf{s}] = [LS(t)] = [LS(t')]$, such that t and t' are Σ-terms, if u is a least general generalizer of t and t' modulo E, then there is a derivation $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$ using the inference rules of Figures 1 and 3, such that u and $(x{:}[\mathsf{s}])\theta$ are equivalent modulo renaming and modulo E.*

Our algorithm straightforwardly terminates for FVP theories whose generalization type is finitary, as illustrated in the following example.

Example 3. Consider an equational theory with one sort s and equations $f(a) = b$ and $f(c) = d$. For the generalization problem $b \overset{x:[\mathsf{s}]}{\triangleq} d$, the trivial generalizer x can be obtained by applying the **Solve** rule but the least general equational generalizer given by $\{x \mapsto f(y)\}$ comes from applying the **Variant** rule (which can be applied only once).

Obviously, termination does not generally hold for FVP theories as witnessed by

Example 4. Consider an equational theory with one sort s and equations $f(a) = a$ and $f(b) = b$. For the generalization problem $a \overset{x:[\mathsf{s}]}{\triangleq} b$, there is an infinite number of increasingly less general generalizers $x_1{:}\mathsf{s}$, $f(x_2{:}\mathsf{s})$, $f(f(x_3{:}\mathsf{s}))$, ..., which can be computed by nondeterministically choosing between the **Solve** and the **Variant** rules at each generalization step. We note that the considered theory is Type 0 (nullary) yet being FVP.

Provided the generalization algorithm terminates for a given problem, strong correctness and completeness directly follow after applying the minimization post-processing.

Theorem 3 (Strong correctness and completeness). *Given a kind-completed, order-sorted equational FVP theory (Σ, E) and a generalization problem $\Gamma = t \overset{x:[\mathsf{s}]}{\triangleq} t'$, with $[\mathsf{s}] = [LS(t)] = [LS(t')]$, such that t and t' are Σ-terms, If the equational generalization algorithm terminates, the minimization post-processing delivers a set of least general equational generalizers for (Σ, E) and Γ.*

5 Conclusion

Computing generalizers is relevant in a wide spectrum of automated reasoning areas where analogical reasoning and inductive inference are needed. We believe

that the equational least general generalization algorithm in this paper opens up a wealth of new applications in many areas where symbolic reasoning modulo equations is convenient. Some key results of this paper can be summarized as follows: (i) anti-unification can be nullary for equational theories that satisfy FVP; (ii) consequently, our complete equational generalization procedure is not in general terminating; (iii) if the procedure stops for a given problem, then the problem has a finite (possibly singleton) minimal complete set of generalizers, and this set can be computed by the subsequent minimization step.

We have formally established the correctness and completeness of our algorithm, while thanks to the minimization post-processing, minimality follows by construction when the algorithm terminates. Similarly to the dual problem of most general E-unification, there are many theories for which least general generalization is nullary (see [7]) and termination is difficult to achieve without quite demanding conditions such as requiring that each E-equivalence class is finite. Actually, our algorithm does not terminate even for theories that satisfy the FVP, as witnessed by Example 4. As future work, we plan to ascertain suitable requirements that may ensure termination of our equational least general generalization algorithm for a wide class of theories.

We are currently developing a prototype implementation of our method, and we plan to develop suitable strategies to boost performance of the tool. We also plan to extend our generic algorithm in order to support equational theories that may contain algebraic axioms such as (A), (C), and (U) following the modular methodology we formalized in [1,2].

References

1. Alpuente, M., Escobar, S., Espert, J., Meseguer, J.: A Modular Order-Sorted Equational Generalization Algorithm. Information and Computation **235**, 98–136 (2014)
2. Alpuente, M., Escobar, S., Meseguer, J., Sapiña, J.: Order-sorted Equational Generalization Algorithm Revisited. Annals of Mathematics and Artificial Intelligence **90**(5), 499–522 (2022). https://doi.org/10.1007/s10472-021-09771-1
3. Armengol, E.: Usages of Generalization in Case-Based Reasoning. In: Proceedings of the 7th International Conference on Case-Based Reasoning (ICCBR 2007). Lecture Notes in Computer Science, vol. 4626, pp. 31–45. Springer (2007). https://doi.org/10.1007/978-3-540-74141-1_3
4. Bader, J., Scott, A., Pradel, M., Chandra, S.: Getafix: Learning to fix bugs automatically. Proc. ACM Program. Lang. **3**(OOPSLA), 159:1–159:27 (2019)
5. Burghardt, J.: E-Generalization using Grammars. Artif. Intell. **165**(1), 1–35 (2005)
6. Cerna, D.M., Kutsia, T.: Idempotent Anti-unification. ACM Transactions on Computational Logic **21**(2), 10:1–10:32 (2020)
7. Cerna, D.M., Kutsia, T.: Unital Anti-Unification: Type and Algorithms. In: Proceedings of the 5th IARCS International Conference on Formal Structures for Computation and Deduction (FSCD 2020). Leibniz International Proceedings in Informatics (LIPIcs), vol. 167, pp. 26:1–26:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.FSCD.2020.26

8. Clavel, M., Durán, F., Eker, S., Escobar, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Rubio, R., Talcott, C.: Maude Manual (Version 3.2.1). Tech. rep., SRI International Computer Science Laboratory (2022), available at: `http://maude.cs.illinois.edu`
9. Comon-Lundh, H., Delaune, S.: The Finite Variant Property: How to Get Rid of Some Algebraic Properties. In: Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA 2005). Lecture Notes in Computer Science, vol. 3467, pp. 294–307. Springer (2005). https://doi.org/10.1007/978-3-540-32033-3_22
10. Durán, F., Eker, S., Escobar, S., Martí-Oliet, N., Meseguer, J., Talcott, C.: Built-in Variant Generation and Unification, and their Applications in Maude 2.7. In: Proceedings of the 8th International Joint Conference on Automated Reasoning (IJCAR 2016). Lecture Notes in Computer Science, vol. 9706, pp. 183–192. Springer (2016)
11. Escobar, S., Sasse, R., Meseguer, J.: Folding Variant Narrowing and Optimal Variant Termination. The Journal of Logic and Algebraic Programming **81**(7–8), 898–928 (2012). https://doi.org/10.1016/j.jlap.2012.01.002
12. Goguen, J.A., Meseguer, J.: Order-sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. Theoretical Computer Science **105**, 217–273 (1992)
13. Kirbas, S., Windels, E., McBello, O., Kells, K., Pagano, M.W., Szalanski, R., Nowack, V., Winter, E.R., Counsell, S., Bowes, D., Hall, T., Haraldsson, S., Woodward, J.R.: On The Introduction of Automatic Program Repair in Bloomberg. IEEE Softw. **38**(4), 43–51 (2021)
14. Kutsia, T., Levy, J., Villaret, M.: Anti-unification for Unranked Terms and Hedges. J. Autom. Reason. **52**(2), 155–190 (2014)
15. Meseguer, J.: Conditional Rewriting Logic as a Unified Model of Concurrency. Theoretical Computer Science **96**(1), 73–155 (1992). https://doi.org/10.1016/0304-3975(92)90182-F
16. Meseguer, J.: Membership Algebra as a Logical Framework for Equational Specification. In: Proceedings of the 12th International Workshop on Algebraic Development Techniques (WADT 1997). Lecture Notes in Computer Science, vol. 1376, pp. 18–61. Springer (1997). https://doi.org/10.1007/3-540-64299-4_26
17. Meseguer, J.: Strict Coherence of Conditional Rewriting Modulo Axioms. Theoretical Computer Science **672**, 1–35 (2017). https://doi.org/10.1016/j.tcs.2016.12.026
18. Muggleton, S.: Inductive Logic Programming: Issues, Results and the Challenge of Learning Language in Logic. Artificial Intelligence **114**(1), 283–296 (1999)
19. Ontañón, S., Plaza, E.: Similarity Measures over Refinement Graphs. Machine Learning **87**(1), 57–92 (2012)
20. Plotkin, G.D.: A Note on Inductive Generalization. Machine Intelligence **5**, 153–163 (1970)
21. Reynolds, J.C.: Transformational Systems and the Algebraic Structure of Atomic Formulas. Machine Intelligence **5**, 135–151 (1970)
22. Talcott, C.: Pathway Logic. In: Proceedings of the 8th International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM 2008). Lecture Notes in Computer Science, vol. 5016, pp. 21–53. Springer (2008)
23. TeReSe: Term Rewriting Systems. Cambridge University Press (2003). https://doi.org/10.1017/S095679680400526X

A An Application of Equational Generalization to a Biological Domain

In this section, we show how our anti-unification methodology can be productively used to analyze biological systems, e.g., to extract similarities and pinpoint discrepancies between two cell models that express distinct cellular states. To illustrate our example, we consider cell states that appear in the MAPK (Mitogen-Activated Protein Kinase) metabolic pathway that regulates growth, survival, proliferation, and differentiation of mammalian cells.

Our cell formalization is inspired by and slightly modifies the data structures used in Pathway Logic (PL) [22] —a symbolic approach to the modeling and analysis of biological systems that is implemented in Maude. Specifically, a cell state can be specified as a typed term as follows.

We use sorts to classify cell entities. The main sorts are `Chemical`, `Protein`, and `Complex`, which are all subsorts of sort `Thing`, which specifies a generic entity. Cellular compartments are identified by sort `Location`, while `Modifier` is a sort that is used to identify post-transactional protein modifications, which are defined by the operator "`[-]`" (e.g., the term `[EgfR - act]` represents the Egf (epidermal growth factor) receptor in an active state). We use the following equations to model modifications of an element `p` of sort `Thing`. Modifications may involve relocation of a chemical, phosphorilation of a protein or the activation of a receptor.

```
eq phosphorilate(p:Thing, X:Modifier) = [ p:Thing - X:Modifier ] .
eq relocate(p:Thing, reloc) = [ p:Thing - reloc ] .
eq activate(p:Thing, act) = [ p:Thing - act ] .
```

A complex is a compound element that is specified by means of the operator "`<=>`", which combines generic entities together.

Now, a *cell state* is represented by a term of the form `[cellType | locs]`, where `cellType` specifies the cell type [1] and `locs` is a list of cellular compartments (or locations). Each location is modeled by a term of the form `{ locName | comp }`, where `locName` is a name identifying the location (e.g., `CLm` represents the cell membrane location), and `comp` is a list that specifies the entities included in that location.

Example 5. The term c_1

```
[ mcell | { Clc | Gab1 relocate(Grb2,reloc) Plcg Sos1 },
          { CLm | EgfR PIP2},
          { CLi | [Src - Yphos] [Hras - GDP] } ]
```

models a cell state of the MAPK pathway with three locations: the cytoplasm (`CLi`) includes four proteins `Gab1`, `Grb2` (which has been relocated), `Plcg`, and `Sos1`; the membrane (`CLm`) includes the receptor `EgfR` and the chemical `PIP2`;

[1] To simplify the exposition, we only consider mammalian cells denoted by the constant `mcell`.

the membrane interior (`CLi`) includes the proteins `Hras` (modified by `GDP`) and the protein `Src` in a phosphorilated state generated by the `Yphos` modifier.

In this scenario, anti-unification can be used to compare two cell states, c_1 and c_2. Indeed, any solution for the problem of generalizing c_1 and c_2 is a term whose non-variable part represents the common cell structure shared by c_1 and c_2, while its variables highlight discrepancy points where the two cell states differ.

Example 6. Consider the problem of generalizing the cell state of Example 5 and the following MAPK cell state c_2

```
[ mcell | { CLc | Gab1 [Grb2 - reloc] Plcg Sos1 },
          { CLm | Egf <=> activate(EgfR, act) PIP2 },
          { CLi | [Src - Tphos] [Hras - GDP] } ]
```

For instance, we can compute the following least general generalizer

```
[ mcell | { CLc | Gab1 [Grb2 - reloc] Plcg Sos1 },
          { CLm | X1:Thing PIP2 },
          { CLi | phosphorilate(Src, X2:Modifier) [Hras - GDP] } ]
```

where `X1:Thing` and `X2:Modifier` are variables. Each variable in the computed lgg detects a discrepancy between the two cell states. The variable `X1:Thing` represents a generic entity that abstracts the status of the receptor `EgfR` in the membrane location `CLm` of the two cells. That is, c_1's membrane includes the (inactive) receptor `EgfR`, whereas c_2's membrane contains the complex `Egf <=> [EgfR - act]` that activates the receptor `EgfR` and binds it to the ligand `Egf` to start the metabolic process. The variable `X2:NModifier` generalizes two phosphorilated states (i.e., `Yphos` and `Tphos`) of the protein `Src` obtained by two distinct phosphorilation modifiers. Note that the computed genralization introduces the partially instantiated function call `phosphorilate(Src, X2:Modifier)` to represent a generic phosphorilation for the protein `Src`.

$$\langle g(f(a,c),a) \overset{w}{\triangleq} g(c,b) \mid \emptyset \mid id \rangle$$

Apply **Decompose**

$$\langle f(a,c) \overset{w_1}{\triangleq} c \wedge a \overset{w_2}{\triangleq} b \mid \emptyset \mid \theta_1 \rangle \text{ with } \theta_1 = \{w \mapsto g(w_1, w_2)\}$$

Apply **Variant** $(w_{12}, \{w_{11} \mapsto e\}) \in [\![f(w_{11}, w_{12})]\!]_E$

$f(a,c)$ is an instance of $f(w_{11}, w_{12})$

c is an instance of w_{12}

$(f(a,c) \overset{w_1}{\triangleq} c) =_E (f(a,c) \overset{w_1}{\triangleq} f(e,c))$

$$\langle a \overset{w_{11}}{\triangleq} e \wedge c \overset{w_{12}}{\triangleq} c \wedge a \overset{w_2}{\triangleq} b \mid \emptyset \mid \theta_1\theta_2 \rangle \text{ with } \theta_2 = \{w_1 \mapsto f(w_{11}, w_{12})\}$$

Apply **Solve**

$$\langle c \overset{w_{12}}{\triangleq} c \wedge a \overset{w_2}{\triangleq} b \mid a \overset{w_{11}}{\triangleq} e \mid \theta_1\theta_2 \rangle$$

Apply **Decompose**

$$\langle a \overset{w_2}{\triangleq} b \mid a \overset{w_{11}}{\triangleq} e \mid \theta_1\theta_2\theta_3 \rangle \text{ with } \theta_3 = \{w_{12} \mapsto c\}$$

Apply **Variant**

$(w_{21}, \{w_{22} \mapsto e\}) \in [\![f(w_{21}, w_{22})]\!]_E$

a is an instance of w_{21}

$(w_{22}, \{w_{21} \mapsto e\}) \in [\![f(w_{21}, w_{22})]\!]_E$

b is an instance of w_{22}

$(a \overset{w_2}{\triangleq} b) =_E (f(a,e) \overset{w_2}{\triangleq} f(e,b))$

$$\langle a \overset{w_{21}}{\triangleq} e \wedge e \overset{w_{22}}{\triangleq} b \mid a \overset{w_{11}}{\triangleq} e \mid \theta_1\theta_2\theta_3\theta_4 \rangle \text{ with } \theta_4 = \{w_2 \mapsto f(w_{21}, w_{22})\}$$

Apply **Recover**

$$\langle e \overset{w_{22}}{\triangleq} b \mid a \overset{w_{11}}{\triangleq} e \mid \theta_1\theta_2\theta_3\theta_4\theta_5 \rangle \text{ with } \theta_5 = \{w_{21} \mapsto w_{11}\}$$

Apply **Solve**

$$\langle \emptyset \mid a \overset{w_{11}}{\triangleq} e \wedge e \overset{w_{22}}{\triangleq} b \mid \theta_1\theta_2\theta_3\theta_4\theta_5 \rangle \text{ with } \theta_1\theta_2\theta_3\theta_4\theta_5 = \{w \mapsto g(f(w_{11}, c), f(w_{11}, w_{22}))\}$$

Fig. 4: Computation trace for equational generalization of terms $g(f(a,c),a)$ and $g(c,b)$.