Term-Graph Anti-Unification

Alexander Baumgartner\textsuperscript{1}, Temur Kutsia\textsuperscript{2}, Jordi Levy\textsuperscript{3}, and Mateu Villaret\textsuperscript{4}

\textsuperscript{1}Department of Computer Science (DCC), University of Chile, Santiago, Chile
\textsuperscript{2}Research Institute for Symbolic Computation, Johannes Kepler University Linz, Austria
\textsuperscript{3}Artificial Intelligence Research Institute, Spanish National Research Council, Barcelona, Spain
\textsuperscript{4}Departament d’Informàtica i Matemàtica Aplicada i Estadística, Universitat de Girona, Spain

Abstract

We study anti-unification for possibly cyclic, unranked term-graphs and develop an algorithm, which computes a minimal complete set of least general generalizations for them. For bisimilar graphs the algorithm computes the join in the lattice generated by a functional bisimulation. Besides, we consider the case when the graph edges are not ordered (modeled by commutativity). These results generalize anti-unification for ranked and unranked terms to the corresponding term-graphs, and solve also anti-unification problems for rational terms and dags. Our results open a way to widen anti-unification based code clone detection techniques from a tree representation to a graph representation of the code.

1 Introduction

Term-graphs are rooted, directed, labeled graphs, which may contain cycles. They can be used to represent functional expressions compactly and to process them efficiently with the help of graph transformations. Rewriting with term-graphs has been studied quite intensively, see, e.g., \cite{4,6,14,18,19,26}. Term-graphs can be represented in various ways, for instance, as constraints \cite{6}, hypergraphs \cite{26}, systems of recursion equations \cite{4}, or arrows in a category \cite{14}. With cycles, term-graphs can express infinite terms and can model regular infinite data structures. Some related (not necessarily equivalent) representations, that are widely used in computer science, include dags, $\mu$-terms, control flow graphs, abstract semantic graphs, program dependency graphs, certain kinds of flowcharts, process graphs, etc.

In this paper, we study the anti-unification problem for term-graphs: Given two such graphs $G_1$ and $G_2$ (maybe with cycles), our goal is to find a graph $G$, which is a least general common generalization of $G_1$ and $G_2$. It means, there should exist variable substitutions $\sigma$.
and $\sigma_2$ such that the instances of $G$ with respect to them, i.e., the graphs $G\sigma_1$ and $G\sigma_2$, are equivalent to $G_1$ and $G_2$, respectively.

Our representation of term-graphs follows the approach from [4], based on recursion equations. The difference is that we are not restricted to ranked alphabets. Variadic function symbols are permitted and, to take the advantage of such variadicity, hedge variables are used together with individual variables. The latter stands for single graphs, while the former can be instantiated by hedges (finite sequences) of graphs. The equivalence relation is bisimilarity.

It has already been shown in [21] that anti-unification for unranked finite terms is finitary: There are, in general, finitely many least general generalizations (lggs). The same holds for unranked term-graphs, discussed in this paper. We develop an algorithm, which computes such lggs. Equivalence class of a term-graph with respect to bisimilarity is a complete lattice. For bisimilar terms, our algorithm computes the lgg, which is the join in this lattice.

The intuition behind lggs is that they should contain “maximal similarities” between the input graphs and should abstract differences between them by variables uniformly. While this might sound similar to the problem of computing maximal common subgraphs (mcs) between graphs [22,23], lggs, in general, might contain more edges than mcs’s and also give information about differences, which is usually neglected in mcs’s.

The results reported in this paper extend our previous results for unranked finite terms [8,21] to unranked cyclic graphs. In particular, we extend rigid anti-unification from terms to graphs. Rigid anti-unification is a more efficient version of the unranked anti-unification algorithm, since it computes only certain kind of generalizations. It is guided by a rigidity function, which, essentially, decides which nodes of the input graphs should be retained in the generalization. Rigidity function is a parameter of the algorithm, whose properties (termination, soundness, completeness) are proved for arbitrary values of this parameter.

As special cases of our results, we obtain anti-unification for ranked term-graphs, rational trees, $\mu$-terms, and dags. To the best of our knowledge, generalization for these structures has not been addressed yet in the literature.

Anti-unification has a pretty wide scope of interesting applications. Originally, it was introduced for inductive reasoning [25]. As a method of computing generalizations, variants of anti-unification are important ingredients of techniques and tools that have found applications in various areas of artificial intelligence, machine learning, reasoning, linguistics, proof theory, program synthesis, analysis, transformation, verification, etc. We can not give an exhaustive overview of all related work here. A couple of recent references (motivated by different applications) include, e.g., [1,2,7,9,10,12,17,20,24]. A particularly interesting motivation comes from software code clone detection, where anti-unification has been successfully incorporated at the level of abstract syntax trees [13,15,27]. Our results can serve as a starting point to extend these techniques for graph-based representation of code (e.g., abstract semantic graphs or program dependence graphs) or graph-based languages (e.g., for model transformation). Besides, term-graph anti-unification can be used to construct an index for sets of dags (e.g., substitution tree indexing), which can be useful in declarative programming and reasoning.

The paper is organized as follows: In Sect. 2, we introduce the notions related to unranked term-graphs. Sect. 3 briefly recalls results about term-graph bisimilarity. The notions related to substitutions and generalizations are introduced in Sect. 4. The term-graph generalization algorithm is described in Sect. 5, and its extension to commutative theory in Sect. 6. Conclusions and the future work are discussed in Sect. 7.

An experimental implementation of our anti-unification algorithm can be accessed online: http://www.risc.jku.at/projects/stout/software/tgau.php.
2 Unranked Cyclic Term-Graphs

We start by defining unranked (possibly infinite) terms. A position \( p \in \mathbb{N}^* \) is a sequence of natural numbers. We use a period to separate numbers in a position, e.g., \( 1.2.3 \). The empty sequence is denoted by \( \epsilon \).

**Definition 1.** Given pairwise disjoint sets of unranked function symbols \( F \) (symbols without fixed arity), term variables \( V_t \), and hedge variables \( V_s \), an unranked term is a partial mapping \( t : \mathbb{N}^* \rightarrow F \cup V_t \cup V_s \) such that

- the domain of \( t \), denoted \( \text{dom}(t) \), is non-empty and prefix-closed (i.e., if \( p_1, p_2 \in \mathbb{N}^* \) and \( p_1. p_2 \in \text{dom}(t) \), then \( p_1 \in \text{dom}(t) \)),
- for all \( p \in \mathbb{N}^* \), if \( t(p) \in F \), then there exists a natural number \( n \geq 0 \) such that \( p.i \in \text{dom}(t) \) for all \( 1 \leq i \leq n \) and \( p.i \notin \text{dom}(t) \) for all \( i > n \),
- for all \( p \in \mathbb{N}^* \), if \( t(p) \in V_t \cup V_s \), then for all \( n \) we have \( p.n \notin \text{dom}(t) \).
- \( t(\epsilon) \notin V_s \).

A term \( t \) is finite if \( \text{dom}(t) \) is a finite set. Otherwise it is infinite. A term is rational if it has finitely many distinct subterms. Hedges are finite (possible empty) sequences of terms and hedge variables. The set of terms (respectively, hedges) over \( F, V_t, \) and \( V_s \) is denoted by \( T(F, V_t, V_s) \) (respectively, \( H(F, V_t, V_s) \)). We use the letters \( f, g, h, a, b, c, \) and \( d \) for function symbols, \( x, y, z \) and \( u \) for term variables, \( X, Y, Z \), and \( U \) for hedge variables, \( \chi, \nu, \upsilon \) and \( \omega \) for a term variable or a hedge variable, \( t \) and \( r \) for terms, \( s \) and \( q \) for a hedge variable or a term, and \( \tilde{s} \) and \( \tilde{q} \) for hedges. The empty hedge is denoted by \( \epsilon \). Given a sequence \( \tilde{s} \), the \( i \)th element of \( \tilde{s} \) is denoted by \( \tilde{s}|_i \). Furthermore, \( \tilde{s}|_i^j \) denotes the subsequence between the positions \( i \) and \( j \) where \( i < j \), that is, \( \tilde{s}|_{i+1}, \ldots, \tilde{s}|_{j-1} \). Unranked terms (resp. hedges) can be naturally represented as unranked trees (resp. forests).

**Example 2.1.** In Fig. 1 we visualize three examples of a finite, infinite non-rational, and infinite rational terms in form of trees. The triangles represent infinite subtrees:

![Term Examples](image)

Figure 1: Unranked terms and their tree representations.

For a term \( t \), we denote by \( V_t(t), V_s(t) \) and \( V(t) \) respectively the sets of term variables, hedge variables, and all variables occurring in \( t \). The notation extends to hedges as well.

Now we define unranked cyclic term-graphs with the help of recursion equations. We start with a very general notion of a system of recursion equations and subsequently impose restrictions to get to the interesting concept.
Definition 2. A system of recursion equations over $\mathcal{F}$, $\mathcal{V}_t$, and $\mathcal{V}_s$ is a set of equations \( \{x_1 \doteq t_1, \ldots, x_n \doteq t_n, X_1 \doteq s_1, \ldots, X_m \doteq s_m\} \), where for all \( i, j, 1 \leq i < j \leq n, x_i \neq x_j \), all \( t_i \)'s are finite terms, for all \( i, j, 1 \leq i < j \leq m, X_i \neq X_j \), and \( s_i \)'s are hedges consisting of finite terms or hedge variables. The variables \( x_1, \ldots, x_n, X_1, \ldots, X_m \) are called recursion variables. They are bound in the system. All other variables occurring in the system are free.

We will use different notation for free and bound variables in systems of recursion equations, writing the latter in bold font. One recursion variable (usually, the leading variable of the first equation) is a designated one and we call it the root of the system $\Gamma$, denoted by $\text{root}(\Gamma)$. It is always a term variable.

A recursion variable $v$ is reachable from a recursion variable $\chi$ in a system $\Gamma$ if $\Gamma$ contains an equation of the form $\chi \doteq s \in \Gamma$ and either $v \in V(s)$, or $v$ is reachable from some recursion variable $u \in V(s)$. In particular, we say that a hedge variable $Y$ is horizontally reachable from a hedge variable $X$ in $\Gamma$, if $\Gamma$ contains an equation $X \doteq s$ such that either $s$ has the form $(\tilde{s}_1, Y, \tilde{s}_2)$, or it has the form $(\tilde{s}_1, Z, \tilde{s}_2)$ and $Y$ is horizontally reachable from $Z$. An equation is called useless in $\Gamma$ if its leading recursion variable is not reachable from $\text{root}(\Gamma)$.

A system $\Gamma$ is called horizontally bounded if no hedge variable is reachable from itself in $\Gamma$, i.e., $\Gamma$ contains no horizontal cycle.\(^1\) For instance, \( \{x \doteq f(x), X \doteq (x, Y)\} \) is a horizontally bounded system, while \( \{x \doteq f(x), X \doteq (x, X)\} \) is not.

We do not distinguish between two systems of recursion equations if they differ from each other only by renaming of bound variables.

A system of recursion equations is called flat, if the equations have one of the following three possible forms: \( x \doteq f(X_1, \ldots, X_n), x \doteq u \) where $u$ is a free or bound term variable, and $X \doteq (v_1, \ldots, v_n)$ where $n \geq 0$ and each $v_i$ is a free or bound term or hedge variable.

A system of recursion equations $\Gamma$ is in canonical form if it does not contain useless equations and each equation in $\Gamma$ has one of the following forms:

- $x \doteq f(X_1, \ldots, X_n)$, where the $X$'s are (not necessarily distinct) recursion variables, or
- $x \doteq y$, where $y$ is a free variable, or
- $X \doteq Y$, where $Y$ is a free variable.

For instance, \( \{x \doteq f(y, X, X), y \doteq g(x), X \doteq Y\} \) and \( \{x \doteq f(y, z, z), y \doteq g(x), z \doteq a\} \) are in canonical form, while \( \{x \doteq f(g(x), X), X \doteq Y\} \), \( \{x \doteq f(y, X), y \doteq g(x)\} \), \( \{x \doteq f(y, X), y \doteq g(x), X \doteq Y, Y \doteq Z\} \) are not.

Every canonical system is flat and horizontally bounded. On the other hand, each flat horizontally bounded system can be transformed to the canonical form by performing the following canonicalization steps as long as possible:

- Remove useless equations.
- Remove trivial equations of the form $x \doteq y$ and replace all occurrences of $x$ by $y$. If $y = x$, then replace the equation by $x \doteq \bullet$, where $\bullet$ is some predefined constant from $\mathcal{F}$.
- Replace equations of the form $X \doteq (v_1, \ldots, v_n)$, $n > 1$, where $v$’s are free or bound term or hedge variables, by $n$ new equations $Y_i \doteq v_i$, $1 \leq i \leq n$, where $Y_i$’s are fresh hedge variables, and replace each occurrence of $X$ by $(Y_1, \ldots, Y_n)$.

\(^1\)Systems that are not horizontally bounded can be used to define cyclic term-graphs where cycles are formed both vertically and horizontally. Such term-graphs can model infinitely branching trees of infinite depth. These structures go beyond the scope of this paper.
• Replace equations of the form $X \doteq u$ by $x \doteq u$ and replace each occurrence of $X$ by $x$, where $u$ is a free or bound term variable and $x$ is a fresh term variable.

• Remove trivial equations of the form $X \doteq Y$ and replace all occurrences of $X$ by $Y$.

• Remove equations of the form $X \doteq \epsilon$ and remove each occurrence of $X$.

Essentially, this canonicalization extends the canonicalization from [4] by four steps dealing with hedge variables. These steps split each equation of the form $X \doteq (s_1,\ldots,s_n)$ into $n$ new equations (one for each $s_i$), and, eventually only those are retained for which $s_i$ is a free variable. The bound $s_i$'s at the end replace their leading recursion variables.

Intuitively, canonical systems of recursion equations can be naturally represented by graphs: The nodes will be the recursion variables; a node $x$ will be connected to a node $\chi$ by an edge if the system contains an equation $x \doteq f(\ldots,\chi,\ldots)$; each node $x$ will have a label $f$ for an equation $x \doteq f(\ldots)$ or the label $y$ for an equation $x \doteq y$, each node $X$ will have a label $Y$ for an equation $X \doteq Y$. Every node is reachable from the root. Cycles and sharings are defined by the occurrences of recursion variables. This intuition justifies the definition:

**Definition 3.** A **term-graph** is a system of recursion equations in canonical form.

**Example 2.2.** We show some term-graphs and their defining recursion equations.

1. Graph: $f \quad x \quad y$
   Equations: $x \doteq f(y, x)$
   $y \doteq a$

2. Graph: $\quad f \quad x \quad y$
   $g \quad y$
   Equations: $x \doteq f(x, y)$
   $y \doteq g(y, x)$

3. Graph: $\quad u \quad x \quad y$
   $f \quad z$
   Equations: $y \doteq f(u, z)$
   $z \doteq f(u, y)$
   $u \doteq x$

A flat horizontally bounded system and its canonical form have the same (possibly infinite) term unwinding. In the rest of the paper we consider only canonical systems of recursion equations. The words “system of recursion equations” and “(term)-graphs” will be used interchangeably. The letter $G$ will be used to denote term-graphs.

Given a term-graph $G$ and an equation $x \doteq t$, the subgraph of $G$ rooted at $x$ is the set $\text{subgraph}(G, x) = \{ x \doteq t \} \cup \{ x \doteq s \mid \chi \doteq s \in G, \text{ where } \chi \text{ is reachable from } x \}$, where $\chi$ is reachable from $x$. Obviously, $\text{subgraph}(G, \text{root}(G)) = G$.

The set of nodes of a term-graph $G$ is denoted by $\text{nodes}(G)$. If $x \in \text{nodes}(G)$ and $v$ is its $i$th successor (i.e., $x \doteq t \in G$ for some $t$ and $v$ is $i$th argument of $t$), we will write $x \rightarrow_i v$. An access path of $v \in \text{nodes}(G)$ is a possibly empty finite sequence of positive natural numbers $i_1, \ldots, i_j$ such that there exist $\chi_1, \ldots, \chi_{j-1} \in \text{nodes}(G)$ with $\text{root}(G) \rightarrow_{i_1} \chi_1 \rightarrow_{i_2} \cdots \rightarrow_{i_{j-1}} \chi_{j-1} \rightarrow_{i_j} v$. 5
A node may have several access paths. The set of all access paths of a node \( \chi \) is denoted by \( \text{acc}(\chi) \).

We will also consider term-graph hedges, defined analogously to hedges: they are finite, possibly empty sequences of term-graphs and hedge variables. We will use \( \mathcal{G} \) to denote them.

### 3 Bisimilarity Relation

It is straightforward to adapt the notions of bisimulation and bisimilarity [4] to our graphs:

**Definition 4.** Let \( \Gamma_1 = \{x_1 \doteq s_1, \ldots, x_n \doteq s_n\} \) and \( \Gamma_2 = \{v_1 \doteq q_1, \ldots, v_m \doteq q_m\} \) be two systems of recursion equations. Then \( R \) is a bisimulation from \( \Gamma_1 \) to \( \Gamma_2 \) iff

- \( R \) is a binary relation with the domain \( \{x_1, \ldots, x_n\} \) and codomain \( \{v_1, \ldots, v_m\} \).
- The roots of \( \Gamma_1 \) and \( \Gamma_2 \) are related: \( x_1 R v_1 \).
- If \( x_i R v_j \), \( x_i \doteq l_1(x'_{1i}, \ldots, x'_{ki}) \in \Gamma_1 \), \( k_i \geq 0 \), and \( v_j \doteq l_2(v'_{1j}, \ldots, v'_{kj}) \in \Gamma_2 \), \( k_j \geq 0 \), then \( l_1 = l_2, k_i = k_j \), and \( x_{nu} R v_u \) for all \( 1 \leq u \leq k_i \). (It applies also when \( l_1 \) and \( l_2 \) are variables: In this case \( k_i = k_j = 0 \).)

In short, bisimulation means that the roots are related, related nodes have the same label, and their successor nodes are again related.

**Definition 5.** Two graphs are bisimilar, if there exists a bisimulation from one to another.

Bisimilarity is an equivalence relation, see, e.g., [4]. We write \( \mathcal{G}_1 \sim \mathcal{G}_2 \) if \( \mathcal{G}_1 \) and \( \mathcal{G}_2 \) are bisimilar, and \( \mathcal{G}_1 \not\succeq \mathcal{G}_2 \) if there exists a functional bisimulation from \( \mathcal{G}_1 \) to \( \mathcal{G}_2 \) (i.e., a bisimulation which is a function).

Functional bisimulation collapses a graph into a smaller one. For the other way around, one says that the graph gets expanded, copied, unwinded, or unraveled. In [4] it is shown that the equivalence class of a term-graph \( \mathcal{G} \) with respect to bisimilarity is a complete lattice, partially ordered by functional bisimulation. The least upper bound in this lattice is a rational term, denoted by \( \triangle \mathcal{G} \), and the greatest lower bound is a fully collapsed graph, denoted by \( \nabla \mathcal{G} \). Hence, \( \triangle \mathcal{G} \succeq \mathcal{G} \succeq \nabla \mathcal{G} \).

**Example 3.1.** Let \( \mathcal{G} \) be the term graph \( \{x \doteq f(y, z), y \doteq a, z = f(y, x)\} \). Then \( \triangle \mathcal{G} \) is the infinite rational term depicted in Fig. 1 in Example 2.1, and \( \nabla \mathcal{G} \) is the first graph in Example 2.2.

Given a bisimulation relation \( R \) from a term-graph \( \mathcal{G}_1 \) to a term-graph \( \mathcal{G}_2 \), its associated graph \( \mathcal{G}_R^A \) is defined as follows: (i) \( \text{nodes}(\mathcal{G}_R^A) = R, \text{root}(\mathcal{G}_R^A) = (\text{root}(\mathcal{G}_1), \text{root}(\mathcal{G}_2)) \), the label of each \( (x_1, x_2) \in \text{nodes}(\mathcal{G}_R^A) \) is that of \( x_1 \) (which is the same as the label of \( x_2 \); (ii) if \( x_1 \in \text{nodes}(\mathcal{G}_1), x_2 \in \text{nodes}(\mathcal{G}_2), (x_1, x_2) \in R, x_1 \rightarrow_i x'_1 \), and \( x_2 \rightarrow_i x'_2 \), then in \( \mathcal{G}_R^A \) we have \( (x_1, x_2) \rightarrow_i (x'_1, x'_2) \).

### 4 Substitutions and Generalizations

The notions related to substitutions, formulated for finite unranked terms and hedges in [21], can be reused with a slight modification for (possibly) infinite terms and hedges.
A substitution is a mapping from term variables to terms and from hedge variables to hedges, which is the identity almost everywhere. We use the traditional finite set representation of substitutions, writing, e.g., \( \{ x \mapsto f(a, f(a, \ldots)) \} \) for the substitution that maps every variable to itself except \( x \), and \( X \). The lower case Greek letters are used to denote substitutions, with the exception of the identity substitution for which we write \( \text{Id} \). The domain and range of a substitution \( \sigma \) are defined in the usual way: \( \text{dom}(\sigma) = \{ \chi \in V \mid \sigma(\chi) \neq \chi \} \) and \( \text{ran}(\sigma) = \{ \sigma(\chi) \mid \chi \in \text{dom}(\sigma) \} \).

Substitutions can be applied to terms and hedges using the congruences \( \sigma(s_1) \neq \sigma(s_2) \) and \( \sigma(s_1 \cdot s_2) = (\sigma(s_1) \cdot \sigma(s_2)) \). We call \( \sigma(s) \) and \( \sigma(\tilde{s}) \) the instances of respectively \( s \) and \( \tilde{s} \) and use postfix notation to denote them, writing \( s \sigma \) and \( \tilde{s} \sigma \). We also say that \( \tilde{s} \) is more general than \( \tilde{q} \) if \( \tilde{q} \) is an instance of \( \tilde{s} \) and denote this fact by \( \tilde{s} \preceq \tilde{q} \). If \( \tilde{s} \preceq \tilde{q} \) and \( \tilde{q} \subseteq \tilde{q} \), then we write \( \tilde{s} \equiv \tilde{q} \). If \( \tilde{s} \preceq \tilde{q} \) and \( \tilde{q} \not\equiv \tilde{q} \), then we say that \( \tilde{s} \) is strictly more general than \( \tilde{q} \) and write \( \tilde{s} < \tilde{q} \).

The composition of two substitutions \( \sigma \) and \( \vartheta \), written as \( \sigma \vartheta \), is defined as the composition of two mappings: We have \( s(\sigma \vartheta) = (s \sigma) \vartheta \) for all \( s \). A substitution \( \sigma_1 \) is more general than \( \sigma_2 \) with respect to a set of variables \( X \subseteq V \), written \( \sigma_1 \preceq_X \sigma_2 \), if there exists \( \vartheta \) such that \( \chi \sigma_1 \vartheta = \chi \sigma_2 \), for each \( \chi \in X \). The relations \( \equiv \) and \( \preceq \) are extended to substitutions: \( \sigma_1 \equiv_X \sigma_2 \) means \( \sigma_1 \preceq_X \sigma_2 \) and \( \sigma_2 \preceq_X \sigma_1 \), and \( \sigma_1 <_X \sigma_2 \) means \( \sigma_1 \preceq_X \sigma_2 \) and \( \sigma_1 \not\equiv_X \sigma_2 \).

Next we define substitutions directly for term-graphs, i.e., for systems of recursion equations (in canonical form). Instead of writing the whole systems of recursion equations in the range of substitutions, only the roots of the corresponding term-graphs appear there. Hedges variables in the image remain unchanged. For instance, assume the term-graphs \( G_1 \) and \( G_2 \) are given by the systems of recursion equations: \( G_1 = \{ x \mapsto f(y, x), y \mapsto a \} \), and \( G_2 = \{ x \mapsto g(X, x, X), X \mapsto Y \} \). Then the substitution \( \{ x \mapsto f(a, f(a, \ldots)), X \mapsto \epsilon, Y \mapsto (X, g(Y, g(Y, \ldots, Y), Y)) \} \) we considered above can be written as \( \{ x \mapsto \text{root}(G_1), X \mapsto \epsilon, Y \mapsto (X, \text{root}(G_2)) \} \).

To define application of such a substitution to a term-graph, we assume that all term-graphs are in canonical form and the bound variables are appropriately renamed. Let \( \sigma \) be a substitution and \( G \) be a term-graph. Then the term-graph \( \sigma(G) \), the instance of \( G \) under \( \sigma \), is obtained by canonicalizing the following flat horizontally bounded system of recursion equations: \( \{ \chi \mapsto \sigma(\tilde{s}) \mid \chi \mapsto \tilde{s} \in \text{dom}(\sigma) \} \cup G_1 \cup \cdots \cup G_n \), where \( G_1, \ldots, G_n \) are all term-graphs whose roots appear in \( \text{ran}(\sigma) \). Substitution application naturally extends to term-graph hedges.

**Example 4.1.** Let \( G \) be the term-graph:

\[
G = \{ x_0 \mapsto f(x_0, X_1, x_1, X_2, x_2), X_1 \mapsto X, x_1 \mapsto g(X_2, x_2, X_2), X_2 \mapsto Y, x_2 \mapsto x \}.
\]

Let \( \sigma = \{ x \mapsto \text{root}(G_1), X \mapsto (\text{root}(G_2), X), Y \mapsto \epsilon \} \), \( G_1 = \{ y \mapsto f(y) \} \), \( G_2 = \{ z \mapsto a \} \). Then

\[
\sigma(G) = \{ x_0 \mapsto f(x_0, z, X, x_1, z, X, Y), z \mapsto a, Z \mapsto X, x_1 \mapsto g(Y, y), y \mapsto f(y) \}.
\]

The notion of more general term-graphs and term-graph hedges is defined modulo bisimilarity: \( G_1 \) is more general than \( G_2 \), if there is a substitution \( \sigma \) such that \( G_1 \sigma \sim G_2 \). We reuse the symbol \( \preceq \) for this relation over term-graphs and term-graph hedges, and also write \( G_1 \preceq G_2 \) if \( G_1 \preceq G_2 \) and \( G_2 \preceq G_1 \). For the strict part of \( \preceq \) we reuse \( < \). Analogously for substitutions: A substitution over term-graphs \( \sigma_1 \) is more general than a substitution over term-graphs \( \sigma_2 \) with
This set is finite and unique modulo \( \sim \). Also in this case we reuse the \( \preceq \) notation and write \( \sigma_1 \preceq \chi \sigma_2 \) (and similarly for the relations \( \simeq \) and \( \prec \) for substitutions).

A term-graph hedge \( \tilde{G} \) is called a generalization of two term-graph hedges \( \tilde{G}_1 \) and \( \tilde{G}_2 \) if \( \tilde{G} \preceq \tilde{G}_1 \) and \( \tilde{G} \preceq \tilde{G}_2 \). We say that a term-graph \( \tilde{G} \) is a least general generalization (lgg in short) of \( \tilde{G}_1 \) and \( \tilde{G}_2 \) if \( \tilde{G} \) is a generalization of \( \tilde{G}_1 \) and \( \tilde{G}_2 \) and there is no generalization \( \tilde{G}' \) of \( \tilde{G}_1 \) and \( \tilde{G}_2 \) that satisfies \( \tilde{G} \prec \tilde{G}' \). That means, there are no generalizations of \( \tilde{G}_1 \) and \( \tilde{G}_2 \) that are strictly less general than their least general generalization.

An anti-unification triple, AUT in short, is written \( \chi : \tilde{G}_1 \triangleq \tilde{G}_2 \), where \( \chi \) does not occur in \( \tilde{G}_1 \) and \( \tilde{G}_2 \). Intuitively, \( \chi \) is a variable that stands for the most general generalization of \( \tilde{G}_1 \) and \( \tilde{G}_2 \). An anti-unifier of \( \chi : \tilde{G}_1 \triangleq \tilde{G}_2 \) is a substitution \( \sigma \) such that \( \text{dom}(\sigma) \subseteq \{ \chi \} \) and \( \chi \sigma \) is a generalization of both \( \tilde{G}_1 \) and \( \tilde{G}_2 \). An anti-unifier \( \sigma \) of an AUT \( \chi : \tilde{G}_1 \triangleq \tilde{G}_2 \) is least general (or most specific) if there is no anti-unifier \( \vartheta \) of the same problem that satisfies \( \sigma \prec_{\{\chi\}} \vartheta \). If \( \sigma \) is a least general anti-unifier of an AUT \( \chi : \tilde{G}_1 \triangleq \tilde{G}_2 \), then \( \chi \sigma \) is an lgg of \( \tilde{G}_1 \) and \( \tilde{G}_2 \).

A complete set of generalizations of two term-graph hedges \( \tilde{G}_1 \) and \( \tilde{G}_2 \) is a set \( G \) of term-graph hedges that satisfies the properties:

**Soundness:** Each \( \tilde{G} \in G \) is a generalization of both \( \tilde{G}_1 \) and \( \tilde{G}_2 \).

**Completeness:** For each generalization \( \tilde{G}' \) of \( \tilde{G}_1 \) and \( \tilde{G}_2 \), there exists \( \tilde{G} \in G \) such that \( \tilde{G}' \preceq \tilde{G} \).

\( G \) is a minimal complete set of generalizations (mcsg) of \( \tilde{G}_1 \) and \( \tilde{G}_2 \) if, in addition to soundness and completeness, it satisfies also the following property:

**Minimality:** For each \( \tilde{G}_1', \tilde{G}_2' \in G \), if \( \tilde{G}_1' \preceq \tilde{G}_2' \) then \( \tilde{G}_1' = \tilde{G}_2' \).

**Lemma 1.** For any hedges \( \tilde{s} \) and \( \tilde{q} \) there exists their minimal complete set of generalizations. This set is finite and unique modulo \( \simeq \).

**Proof.** Similar to the analogous lemma for hedges with finite terms, see [21]. \( \square \)

**Theorem 1.** For any term-graph hedges \( \tilde{G}_1 \) and \( \tilde{G}_2 \) there exists their minimal complete set of generalizations. This set is finite and unique modulo \( \simeq \) and \( \sim \).

**Proof.** Note that \( \tilde{G} \sim \Delta \tilde{G} \) for all \( \tilde{G} \). Let \( \tilde{G}_1 = (\tilde{G}_1^1, \ldots, \tilde{G}_1^n) \) and \( \tilde{G}_2 = (\tilde{G}_2^1, \ldots, \tilde{G}_2^m) \). By Lemma 1, the hedges \((\Delta \tilde{G}_1^1, \ldots, \Delta \tilde{G}_1^n)\) and \((\Delta \tilde{G}_2^1, \ldots, \Delta \tilde{G}_2^m)\) have a finite minimal complete set of generalizations, unique modulo \( \simeq \). \( \square \)

Our goal is not to compute minimal complete sets of generalizations. We would rather focus on so called rigid generalizations, which we define below. The motivation comes from the experience with finite unranked term anti-unification, where unrestricted mcsg can grow too big and it makes sense to restrict consecutive hedge variables in the generalization. For the details, see [21].

**Definition 6** (Alignment, Rigidity Function). Let \( w_1 \) and \( w_2 \) be strings of symbols. Then the sequence \( a_1[i_1, j_1] \cdots a_n[i_n, j_n] \), for \( n \geq 0 \), is an alignment if \( i \)'s and \( j \)'s are positive integers such that \( 0 < i_1 < \cdots < i_n < |w_1| \) and \( 0 < j_1 < \cdots < j_n < |w_2| \), and \( a_k = w_1[i_k = w_2[j_k] \) for all \( 1 \leq k \leq n \). A rigidity function \( \mathcal{R} \) is a function that returns, for every pair of strings of symbols \( w_1 \) and \( w_2 \), a set of alignments of \( w_1 \) and \( w_2 \).

\(^2\)Note that unrestricted unranked term anti-unification (i.e., without a rigidity function) can be also modeled as associative anti-unification with the unit element. The latter problem has been studied e.g., in [2, 3].
For instance, if $R$ computes the set of all longest common subsequences, then $R(abcda, becad) = \{b, [2, 1]c[3, 2]a[5, 3], b[2, 1]c[3, 2]d[4, 4]\}$.

The \textit{top symbol} of a term is defined as $\text{top}(x) = x$ for any variable $x$, and $\text{top}(f(s)) = f$ for any term $f(s)$. The notion is extended to hedges: $\text{top}(X) = X$ and $\text{top}(s_1, \ldots, s_n) = (\text{top}(s_1), \ldots, \text{top}(s_n))$. If $\{X_1 \vdash s_1, \ldots, X_n \vdash s_n\} \subseteq G$, $n > 0$, then we define $\text{top}(X_1, \ldots, X_n, G)$ as $\text{top}(s_1, \ldots, s_n)$. Moreover, we define $\text{top}(G) = \text{top}($\text{root}(G), G$).

\textbf{Definition 7} \textit{(Commutative Alignment).} Let $w_1$ and $w_2$ be strings of symbols. Then the sequence $a_1[i_1, j_1] \cdots a_n[i_n, j_n]$, for $n \geq 0$, is a \textit{commutative alignment} if $i$'s and $j$'s are positive integers such that $i_k \neq j_l$ for all $k \neq l$, and $j_k \neq j_l$ for all $k \neq l$, and $a_k = w_1[i_k] = w_2[j_k]$ for all $1 \leq k \leq n$.

We denote by $R_C$ the commutative counterpart of a rigidity function $R$ that computes a set of commutative alignments for a pair of strings of symbols.

\textbf{Definition 8} \textit{(R-Generalization).} Given two term-graphs $G_1$ and $G_2$ (without common free and bound variables) and the rigidity function $R$, we say that a term-graph $G$ that generalizes both $G_1$ and $G_2$ is their \textit{generalization with respect to} $R$, or, shortly, an $R$-generalization, if either

1. $R(\text{top}(G_1), \text{top}(G_2)) \in \{\emptyset, \{x \vdash y\}\}$ and $G = \{x \vdash y\}$, where $x$ is a new bound term variable and $y$ is a new free term variable, or
2. $f[1, 1] \in R(\text{top}(G_1), \text{top}(G_2))$ for some $f$ and $G = \{\text{root}(G) \vdash f(\tilde{\chi})\} \cup \bigcup G_1' \cup \cdots \cup G_n'$, where $\tilde{\chi}$ does not contain pairs of consecutive hedge recursion variables.

The sequence $\tilde{\chi}$, the set $\bigcup$, and the graphs $G_1', \ldots, G_n'$ are defined as follows:

For $i = 1, 2$, the original graph $G_i$ contains an equation $\text{root}(G_i) \vdash f(\tilde{\nu}_i)$ and there exists an alignment $g_1[i_1, j_1] \cdots g_n[i_n, j_n] \in R(\text{top}(\tilde{\nu}_1, G_1), \text{top}(\tilde{\nu}_2, G_2))$, satisfying the following conditions:

1. If we remove all hedge recursion variables that occur as elements of $\tilde{\chi}$, we get a sequence of term recursion variables $(x_1, \ldots, x_n)$, such that $x_k = \text{root}(G_k')$ and each $G_k'$ contains an equation of the form $x_k \vdash g_k(\tilde{\nu}_k)$ for all $1 \leq k \leq n$, and
2. For every $1 \leq k \leq n$, there exists a pair of term recursion variables $y_k^1$ and $y_k^2$ such that $\tilde{\nu}_1[i_k] = y_k^1$, $\tilde{\nu}_2[j_k] = y_k^2$, and $G_k'$ is an $R$-generalization of subgraph $(G_1, y_k^1)$ and subgraph $(G_2, y_k^2)$.
3. $\bigcup \{Y_1 \vdash Z_1, \ldots, Y_m \vdash Z_m\}$, where $Y_1, \ldots, Y_m$ are all hedge recursion variables in $\tilde{\chi}$ and $Z_1, \ldots, Z_m$ are new free hedge variables.

\textbf{Example 4.2.} Let $R$ compute the set of all longest common subsequences and let $G_1 = \{x_0 \vdash f(x_1, x_2), x_1 \vdash g(x_2, x_2), x_2 \vdash a\} \text{ and } G_2 = \{y_0 \vdash f(y_1, y_0, y_2, y_0), y_1 \vdash g, y_2 \vdash a\}$. The term graph $\{z_0 \vdash f(z_1, Z_1, z_2, Z_1), z_1 \vdash g(Z_2), z_2 \vdash a, Z_1 \vdash Z_1, Z_2 \vdash Z_2\}$ is an $R$-generalization of $G_1$ and $G_2$ while $\{z_0 \vdash f(z_1, Z_1, z_2, Z_1), z_1 \vdash g(Z_3, Z_2), z_2 \vdash a, Z_1 \vdash Z_1, Z_2 \vdash Z_2\}$ and $\{z_0 \vdash f(z_1, Z_1, z_2, Z_1), z_1 \vdash z, z_2 \vdash a, Z_1 \vdash Z_1\}$ are not.

\section{The Algorithm}

We present our anti-unification algorithm as a rule-based algorithm that works on quadruples $A; S; T; G$, called configurations. Here $A$, $S$, and $T$ are sets of anti-unification triples and $G$ is
a term-graph. The rules transform configurations into configurations. Intuitively, the problem set \( A \) contains AUTs that have not been solved yet, the store \( S \) contains the already solved AUTs, the trail \( T \) keeps track of the names of recursion variables, and \( \mathcal{G} \) is the generalization which becomes more and more specific as the algorithm progresses by applying the rules.

To keep the notation short, in anti-unification triples we only use variables from the graphs to be generalized. Those graphs are not explicitly present in the configurations, but are global parameters, denoted by \( \mathcal{G}_1 \) and \( \mathcal{G}_2 \). For simplicity, we assume that \( \mathcal{G}_1 \) and \( \mathcal{G}_2 \) do not contain free variables. This is not a restriction, because we can replace free variables by new constants, use the algorithm defined below, and in the generalization replace those constants back with variables. (In case of hedge variables, we might need to replace their corresponding generalization term variables by generalization hedge variables.) The rigidity function \( \mathcal{R} \) is yet another global parameter. In the rules below, generalization term-graphs are assumed to be implicitly transformed into the canonical form.

**Step: Simplification Step**

\[
\{ x : y \triangleq z \} \cup A; S; T; \mathcal{G} \Longrightarrow A_0 \cup A; S; T \cup \{ u : y \triangleq z \}; \mathcal{G}\{ x \mapsto u \} \cup \{ u \triangleq t \},
\]

where \( y \equiv \ell(\upsilon) \in \mathcal{G}_1 \), \( z \equiv \ell(\upsilon) \in \mathcal{G}_2 \), \( \ell(1, 1) \in \mathcal{R}(\text{top}(y, \mathcal{G}_1), \text{top}(z, \mathcal{G}_2)) \), \( T \) does not contain an AUT of the form \( \_ : y \triangleq z \), and \( u \) is a fresh recursion term variable. If \( \upsilon = \upsilon = \epsilon \) then \( t = l \) and \( A_0 = \emptyset \), otherwise \( t = l(X) \) and \( A_0 = \{ X : \upsilon \triangleq \upsilon \} \) where \( X \) is a fresh hedge variable.

**Dec-S: Decomposition and Solving**

\[
\{ x : \upsilon \triangleq \upsilon \} \cup A; S; T; \mathcal{G} \Longrightarrow
\]

\[
A \cup \{ y_k : \upsilon|_{i_k} \triangleq \upsilon|_{j_k} \mid 1 \leq k \leq n \};
\]

\[
S \cup \{ Y_0 : \upsilon|_{i_0} \triangleq \upsilon|_{j_0} \} \cup \{ Y_k : \upsilon|_{i_k} \triangleq \upsilon|_{j_k} \mid 1 \leq k \leq n - 1 \} \cup \{ Y_n : \upsilon|_{i_n} \triangleq \upsilon|_{j_n} \};
\]

\[
T; \mathcal{G} \sigma \cup \{ Z_0 = Y_0, \ldots, Z_n = Y_n \},
\]

if \( \mathcal{R}(\text{top}(\upsilon, \mathcal{G}_1), \text{top}(\upsilon, \mathcal{G}_2)) \) contains a sequence \( l_1[i_1, j_1] \cdots l_n[i_n, j_n], n > 0 \). The \( y \)'s are fresh term variables, the \( Y \)'s are fresh hedge variables, the \( Z \)'s are fresh recursion hedge variables, and the substitution is \( \sigma = \{ X \mapsto (Z_0, y_1, Z_1, \ldots, Z_{n-1}, y_n, Z_n) \} \). For each \( 1 \leq i \leq n \), if the new AUT has the form \( Y_i : \epsilon \triangleq \epsilon \), then it is not added to \( S \) and \( Z_i \) does not appear in \( \sigma \).

**Solve: Solving**

\[
\{ x : \upsilon \triangleq \upsilon \} \cup A; S; T; \mathcal{G} \Longrightarrow A \cup \{ x : \upsilon \triangleq \upsilon \};\mathcal{G}\{ \chi \mapsto \omega \} \cup \{ \omega \triangleq \chi \},
\]

if \( \mathcal{R}(\text{top}(\upsilon, \mathcal{G}_1), \text{top}(\upsilon, \mathcal{G}_2)) = \emptyset \) or \( \mathcal{R}(\text{top}(\upsilon, \mathcal{G}_1), \text{top}(\upsilon, \mathcal{G}_2)) = \{ \epsilon \} \). The variable \( \omega \) is a fresh recursion variable. If \( \chi \in \mathcal{V}_t \), then \( \omega \in \mathcal{V}_t \) and if \( \chi \in \mathcal{V}_s \), then \( \omega \in \mathcal{V}_s \).

**Share: Sharing**

\[
\{ x : y \triangleq z \} \cup A; S; \{ u : y \triangleq z \} \cup T; \mathcal{G} \Longrightarrow A; S; \{ u : y \triangleq z \} \cup T; \mathcal{G}\{ x \mapsto u \}.
\]

**Merge: Merging Nodes in the Store**

\[
\emptyset; \{ x_1 : \upsilon \triangleq \upsilon, x_2 : \upsilon \triangleq \upsilon \} \cup S; T; \{ \omega_1 \triangleq \chi_1, \omega_2 \triangleq \chi_2 \} \cup \mathcal{G} \Longrightarrow
\]

\[
\emptyset; S \cup \{ x_1 : \upsilon \triangleq \upsilon \}; T; \mathcal{G}\{ \omega_2 \mapsto \omega_1 \} \cup \{ \omega_1 \triangleq \chi_1 \},
\]

where \( \chi_1, \chi_2 \in \mathcal{V}_t \cup \mathcal{V}_s \) such that if \( \chi_1 \in \mathcal{V}_s \), then \( \chi_2 \not\in \mathcal{V}_t \).

The rules never generate the AUTs of the form \( X : \epsilon \triangleq \epsilon \). To compute \( \mathcal{R} \)-generalizations of \( \mathcal{G}_1 \) and \( \mathcal{G}_2 \), we start with \( \{ x : \text{root}(\mathcal{G}_1) \triangleq \text{root}(\mathcal{G}_2) \}, \emptyset, \emptyset, \{ x \triangleq x \} \) and apply the rules on the selected AUTs in all possible ways. The obtained procedure is denoted by \( \text{Gen}(\mathcal{R}) \).
The notation $\Rightarrow^*$ abbreviates finite (possible empty) sequence of rule applications. If we want to make it clear which rule is used to transform a configuration, we will write the rule name as the index at the arrow like, e.g., $A;S : T; G \Rightarrow_{\text{Step}} A'; S' : T'; G'$ for the transformation with the rule Simplification Step.

**Example 5.1.** Let $\mathcal{R}$ be the longest common subsequence. Then the term-graphs $G_1$ and $G_2$ below have a unique $\mathcal{R}$-lgg $G$:

$G_1 = \{x_0 \triangleright f(x_1, x_2, x_3, x_0, x_2, x_3), x_1 \triangleright g(x_1, x_2), x_2 \triangleright b, x_3 \triangleright a\}$.

$G_2 = \{y_0 \triangleright f(y_1, y_0, y_3), y_1 \triangleright g(y_1, y_2), y_2 \triangleright b, y_3 \triangleright a\}$.

$G = \{z_0 \triangleright f(z_1, z_1, z_0, z_3, Z_1), z_1 \triangleright g(z_1, z_2), Z \triangleright U, z_3 \triangleright a, z_2 \triangleright b\}.$

Graphically:

![Term-graphs](image)

The algorithm $\text{Gen}(\mathcal{R})$ computes $G$, e.g., in the following way:

\[
\{u_0 : x_0 \triangleq y_0\}; \emptyset; \{z_0 \triangleq u_0\} \Rightarrow_{\text{Step}}
\]

\[
\{U_0 : (x_1, x_2, x_3, x_0, x_2, x_3) \triangleq (y_1, y_0, y_3)\}; \emptyset; \{z_0 : x_0 \triangleq y_0\};
\]

\[
\{z_0 \triangleright f(Z_0), Z_0 \triangleright U_0\} \Rightarrow_{\text{Dec-S}}
\]

(Choosing the common subsequence: $g[1, 1]f[4, 2]a[5, 3]$), corresponding to the node pairs $x_1$ and $y_1$, $x_0$ and $y_0$, the second occurrence of $x_3$ and $y_3$.)

\[
\{u_1 : x_1 \triangleq y_1, u_2 : x_0 \triangleq y_0, u_3 : x_3 \triangleq y_3\}; \{U_1 : (x_2, x_3) \triangleq \epsilon, U_2 : (x_2, x_3) \triangleq \epsilon\};
\]

\[
\{z_0 : x_0 \triangleq y_0\};
\]

\[
\{z_0 \triangleright f(u_1, Z_1, u_2, u_3, Z_2), u_1 \triangleright U_1, u_2 \triangleright u_2, u_3 \triangleright u_3, Z_2 \triangleright U_2\} \Rightarrow_{\text{Step}}
\]

\[
\{u_2 : x_0 \triangleq y_0 , u_3 : x_3 \triangleq y_3, U_3 : (x_1, x_2) \triangleq (y_1, y_2)\};
\]

\[
\{U_1 : (x_2, x_3) \triangleq \epsilon, U_2 : (x_2, x_3) \triangleq \epsilon\}; \{z_0 : x_0 \triangleq y_0, z_1 : x_1 \triangleq y_1\};
\]

\[
\{z_0 \triangleright f(z_1, Z_1, u_3, Z_2), z_1 \triangleright g(Z_3), Z_1 \triangleright U_1, u_2 \triangleright u_2, u_3 \triangleright u_3, Z_2 \triangleright U_2, Z_3 \triangleright U_3\} \Rightarrow_{\text{Share}}
\]

\[
\{u_3 : x_3 \triangleq y_3, U_3 : (x_1, x_2) \triangleq (y_1, y_2)\};
\]

\[
\{U_1 : (x_2, x_3) \triangleq \epsilon, U_2 : (x_2, x_3) \triangleq \epsilon\}; \{z_0 : x_0 \triangleq y_0, z_1 : x_1 \triangleq y_1\};
\]

\[
\{z_0 \triangleright f(z_1, Z_1, u_3, Z_2), z_1 \triangleright g(Z_3), Z_1 \triangleright U_1, u_3 \triangleright u_3, Z_2 \triangleright U_2, Z_3 \triangleright U_3\} \Rightarrow_{\text{Step}}
\]

\[
\{U_3 : (x_1, x_2) \triangleq (y_1, y_2)\};
\]

\[
\{U_1 : (x_2, x_3) \triangleq \epsilon, U_2 : (x_2, x_3) \triangleq \epsilon\}; \{z_0 : x_0 \triangleq y_0, z_1 : x_1 \triangleq y_1, z_3 : x_3 \triangleq y_3\};
\]

\[
\{z_0 \triangleright f(z_1, Z_1, u_3, Z_2), z_1 \triangleright g(Z_3), Z_1 \triangleright U_1, z_3 \triangleright a, Z_2 \triangleright U_2, Z_3 \triangleright U_3\} \Rightarrow_{\text{Dec-S}}
\]
(Choosing the common subsequence: \(g[1, 1][2, 2]\), corresponding to the node pairs \(x_1\) and \(y_1\), \(x_2\) and \(y_2\))

\[
\begin{align*}
\{u_4 : & x_1 \triangleq y_1, u_5 : x_2 \triangleq y_2; \\
& \{U_1 : (x_2, x_3) \triangleq \epsilon, U_2 : (x_2, x_3) \triangleq \epsilon; \{z_0 : x_0 \triangleq y_0, z_1 : x_1 \triangleq y_1, z_3 : x_3 \triangleq y_3; \\
& \{z_0 \triangleq f(z_1, Z_1, z_0, z_3, Z_2), \\
& z_1 \triangleq g(u_4, u_5), Z_1 \triangleq U_1, z_3 \triangleq a, Z_2 \triangleq U_2, u_4 \triangleq u_4, u_5 \triangleq u_5 \} \Rightarrow \text{Share} \}
\end{align*}
\]

\[
\begin{align*}
\{u_5 : & x_2 \triangleq y_2; \\
& \{U_1 : (x_2, x_3) \triangleq \epsilon, U_2 : (x_2, x_3) \triangleq \epsilon; \{z_0 : x_0 \triangleq y_0, z_1 : x_1 \triangleq y_1, z_3 : x_3 \triangleq y_3; \\
& \{z_0 \triangleq f(z_1, Z_1, z_0, z_3, Z_2), \\
& z_1 \triangleq g(z_1, u_3), Z_1 \triangleq U_1, z_3 \triangleq a, Z_2 \triangleq U_2, u_5 \triangleq u_5 \} \Rightarrow \text{Step} \}
\end{align*}
\]

\[
\begin{align*}
\emptyset : & \{U_1 : (x_2, x_3) \triangleq \epsilon, U_2 : (x_2, x_3) \triangleq \epsilon; \{z_0 : x_0 \triangleq y_0, z_1 : x_1 \triangleq y_1, z_3 : x_3 \triangleq y_3, z_2 : x_2 \triangleq y_2; \\
& \{z_0 \triangleq f(z_1, Z_1, z_0, z_3, Z_2), \\
& z_1 \triangleq g(z_1, z_2), Z_1 \triangleq U_1, z_3 \triangleq a, Z_2 \triangleq U_2, z_2 \triangleq b \} \Rightarrow \text{Merge} \}
\end{align*}
\]

\[
\begin{align*}
\emptyset : & \{U_1 : (x_2, x_3) \triangleq \epsilon; \{z_0 : x_0 \triangleq y_0, z_1 : x_1 \triangleq y_1, z_3 : x_3 \triangleq y_3, z_2 : x_2 \triangleq y_2; \\
& \{z_0 \triangleq f(z_1, Z_1, z_0, z_3, Z_1), z_1 \triangleq g(z_1, z_2), Z_1 \triangleq U_1, z_3 \triangleq a, z_2 \triangleq b \} \Rightarrow \text{Merge} \}
\end{align*}
\]

The obtained generalization is equal to \(G\) modulo renaming variables. The store and the trail suggest how to obtain the original term-graphs from the computed generalization. For instance, to obtain \(G_1\) from \(G\), we just apply the substitution \(\{U \mapsto (x_2, x_3)\}\) to \(G\). In the obtained term-graph we will have \(x_2 \triangleq b\) and \(x_3 \triangleq a\) alongside to \(z_2 \triangleq b\) and \(z_3 \triangleq a\), but it will be bisimilar to \(G_1\).

**Example 5.2.** Let \(G_1 = \{x_0 \triangleq f(x_1, x_2), x_1 \triangleq g(x_0, x_3), x_2 \triangleq a, x_3 \triangleq b\}\) and \(G_2 = \{y_0 \triangleq f(y_1, y_2), y_1 \triangleq h(y_0, y_3), y_2 \triangleq a, y_3 \triangleq b\}\). Then the algorithm ends with \(\emptyset, \{z : x_1 \triangleq y_1\}, \{z_0 : x_0 \triangleq y_0, z_1 : x_1 \triangleq y_1, z_3 : x_3 \triangleq y_3, z_2 : x_2 \triangleq y_2\}\). Obtaining \(G_1\) from the computed generalization can be illustrated as \(\{z_0 \triangleq f(z_1, z_2), z_1 \triangleq z, z_2 \triangleq a\}\) \{\(z \mapsto x_1\) = \(\{z_0 \triangleq f(x_1, z_2), z_2 \triangleq a, x_1 \triangleq g(x_0, x_3), x_0 \triangleq f(x_1, x_2), x_2 \triangleq a, x_3 \triangleq b\} \Rightarrow G_1\).}

**Theorem 2** (Termination). The procedure \(\text{Gen}(\mathcal{R})\) terminates on any input and produces a configuration \(\emptyset; S; T; G\), where \(S\) is irreducible with respect to the merging rule.

**Proof.** Let the size of a hedge, \(size(\tilde{s})\), be the number of symbols in it. the size of an AUT \(x : t_1 \triangleq t_2\) be \(size(t_1) + size(t_2) + 1\), and the size of \(X : \tilde{s_1} \triangleq \tilde{s_2}\) be \(size(\tilde{s_1}) + size(\tilde{s_2}) + 2\). The size of a set of AUTs is the multiset of the sizes of its elements. Then the only rule that increases the size of \(A\) is Step. However, this step can be applied only finitely many times, since each time it strictly decreases the number of unvisited node pairs \((x_1, x_2)\), where \(x_1 \in G_1\) and \(x_2 \in G_2\). Any other rule strictly decreases the size of \(A\) or, in case of \(\text{Merge}\), the size of \(S\). Moreover, \(\text{Merge}\) does not change the size of \(A\). The rule \(\text{Dec-S}\) can introduce only finite branching. Therefore, the algorithm terminates.

**Definition 9.** Given a set \(A\) of AUTs where all the generalization variables are pairwise distinct. We define two substitutions that can be obtained from \(A\):

\[
\sigma_L(A) := \{\chi \mapsto \tilde{v} \mid \chi : \tilde{v} \triangleq \tilde{u} \in A\} \quad \sigma_R(A) := \{\chi \mapsto \tilde{v} \mid \chi : \tilde{v} \triangleq \tilde{u} \in A\}
\]
Lemma 2. Let \( \mathcal{G}_1, \mathcal{G}_2 \) be the two term graphs to be generalized and let \( A; S; T; \mathcal{G} \) be a configuration such that all the generalization variables from \( A, S, T \) are unique among all the other variables from \( A, S, T \), including those occurring in graphs or hedges. Furthermore, let \( \mathcal{G}\sigma_L(T)\sigma_R(S)\sigma_L(A) = \mathcal{G}_1 \) and \( \mathcal{G}\sigma_R(T)\sigma_R(S)\sigma_R(A) = \mathcal{G}_2 \), and let \( \mathcal{G} \) be a rigid generalization of \( \mathcal{G} \) and \( \mathcal{G}_i \) where \( i \in \{1,2\} \).

If \( A; S; T; \mathcal{G} \rightarrow A'; S'; T'; \mathcal{G}' \) is a transformation step applying one of the defined rules then all the generalization variables from \( A', S', T' \) are unique among all the other variables from \( A', S', T' \). Moreover, \( \mathcal{G}'\sigma_L(T')\sigma_L(S')\sigma_L(A') = \mathcal{G}_1 \) and \( \mathcal{G}'\sigma_R(T')\sigma_R(S')\sigma_R(A') = \mathcal{G}_2 \), and \( \mathcal{G}' \) is a rigid generalization of \( \mathcal{G}' \) and \( \mathcal{G}_i \) where \( i \in \{1,2\} \).

Proof. We prove that each rule preserves those properties. We can omit the proof for \( \mathcal{G}'\sigma_R(T')\sigma_R(S')\sigma_R(A') = \mathcal{G}_2 \), since it is equivalent to proving \( \mathcal{G}'\sigma_L(T')\sigma_L(S')\sigma_L(A') = \mathcal{G}_1 \). For the same reason, we omit the proof that \( \mathcal{G}' \) is a rigid generalization of \( \mathcal{G}' \) and \( \mathcal{G}_2 \).

In Step we have two cases, namely (i) \( \nu = \bar{\nu} = \epsilon \), and (ii) \( \nu \neq \epsilon \) or \( \bar{\nu} \neq \epsilon \). We only illustrate the more general case (ii) since the two proofs are largely identical. Therefore, we have \( A = \{ x : y \triangleq z \} \cup \{ X : \bar{\nu} \triangleq \bar{\nu} \}, S = S', T \cup \{ u : y \triangleq z \} = T', \) and \( \mathcal{G}[x \mapsto u] \cup \{ u \triangleq l(X) \} = \mathcal{G}' \), where \( y = l(\nu) \in \mathcal{G}_1, z = l(\bar{\nu}) \in \mathcal{G}_2 \) and \( u, X \) are fresh. Since \( u, X \) are fresh, all the generalization variables from \( A', S', T' \) are still unique among all the other variables from \( A', S', T' \). Obviously, \( \mathcal{G}\sigma_L(T)\sigma_L(S)\sigma_L(A) = \mathcal{G}\sigma_L(T)\sigma_L(S')\sigma_L(A) \{ x : y \triangleq z \} \cup \{ X : \bar{\nu} \triangleq \bar{\nu} \} \). From the uniqueness of \( x \), and by definition of substitution application follows that \( \mathcal{G}_1 = \{ x \mapsto y \}\sigma_L(T)\sigma_L(S)'\sigma_L(A) \{ X : \bar{\nu} \triangleq \bar{\nu} \} = \{ x \mapsto y \}\sigma_L(T)\sigma_L(S)'\sigma_L(A) \{ X : \bar{\nu} \triangleq \bar{\nu} \} \}. \) From the uniqueness of \( X \) follows \( \mathcal{G}_1 = \{ x \mapsto y \} \cup \{ y \triangleq l(X) \} = \{ x \mapsto y \} \cup \{ y \triangleq l(X) \} = \{ x \mapsto y \} \sigma_L(T)\sigma_L(S)'\sigma_L(A) \). Finally, from the uniqueness of \( u \) follows \( \mathcal{G}_1 = \{ x \mapsto y \} \cup \{ y \triangleq l(X) \} \sigma_L(T)\sigma_L(S)'\sigma_L(A) = \{ x \mapsto y \} \sigma_L(T)\sigma_L(S)'\sigma_L(A) = \{ x \mapsto y \} \sigma_L(T)\sigma_L(S)'\sigma_L(A) \}. \)

Since Step can’t lead to consecutive hedge variables and \( l[1,1] \in \mathcal{R}(\text{top}(\nu, G_1), \text{top}(z, G_2)) \), it follows that \( \mathcal{G}' \) is a rigid generalization of \( \mathcal{G}' \) and \( \mathcal{G}_1 \).

Now we analyze Dec-S, which is a bit more involved. We have \( A = \{ x : \bar{\nu} \triangleq \nu \} \cup \{ y : \nu \triangleq \nu \} \cup \{ y_k : \nu \triangleq \nu \} \cup \{ y_k : \nu \triangleq \nu \} \), \( S \cup \{ Y_0 : \nu \triangleq \nu \} \cup \{ Y_k : \nu \triangleq \nu \} \) and \( \mathcal{G}\sigma \cup \{ Z_0 \triangleq Y_0, \ldots, Z_n \triangleq Y_n \} = \mathcal{G}' \), where \( \mathcal{R}(\text{top}(\nu, \mathcal{G}_1), \text{top}(\nu, \mathcal{G}_2)) \) contains a sequence \( l[i_1, j_1] \cdots l[i_n, j_n], n > 0 \), the left \( \nu \)'s, \( \bar{\nu} \)'s, and \( \nu \)'s are fresh, and \( \sigma = \{ X \mapsto (Z_0, y_1, Z_1, \ldots, Z_{n-1}, y_n, Z_n) \} \). Since all the variables introduced by the transformation are fresh, all the generalization variables from \( A', S', T' \) are still unique. We get \( \mathcal{G}\sigma_L(T)\sigma_L(S)\sigma_L(A) = \mathcal{G}\sigma_L(T)\sigma_L(S')\sigma_L(A) \{ \{ Y_0 : \nu \triangleq \nu \} \cup \{ Y_k : \nu \triangleq \nu \} \cup \nu \} \). By uniqueness of \( X \), follows \( \mathcal{G}_1 = \{ x \mapsto \nu \} \sigma_L(T)\sigma_L(S)'\sigma_L(A) \{ \{ Y_0 : \nu \triangleq \nu \} \cup \{ Y_k : \nu \triangleq \nu \} \cup \nu \} \). Now observe that \( \mathcal{G}\sigma \cup \{ Y_0 \triangleq Y_0, \ldots, Y_n \triangleq Y_n \} \) is equivalent to \( \mathcal{G}\sigma \cup \{ Y_0 \triangleq Y_0, \ldots, Y_n \triangleq Y_n \} \) because it is very similar to the case of Step 13.
In Share we have $A = \{x : y \triangleq z\} \cup A'$, $S = S'$, $T = T'$, and $\mathcal{G}\{x \mapsto u\} = \mathcal{G}'$, where $\{u : y \triangleq z\} \in T$. Uniqueness of generalization variables from $A', S', T'$ is obviously maintained. We get $\mathcal{G}\sigma_L(T)\sigma_L(S)\sigma_L(A) = \mathcal{G}\sigma_L(T')\sigma_L(S')\sigma_L(\{x : y \triangleq z\} \cup A') = \mathcal{G}_1$ and by uniqueness of $x$ follows $\mathcal{G}_1 = \mathcal{G}\{x \mapsto y\}\sigma_L(T')\sigma_L(S')\sigma_L(A')$. The trail $\{u : y \triangleq z\} \in T$ tells us that there is already a recursion variable $u$ in $\mathcal{G}$ that represents the node $y$ in $\mathcal{G}_1$. Therefore, instead of substituting $x$ with $y$ we may as well substitute it with $u$. This consideration leads to $\mathcal{G}_1 = \mathcal{G}\{x \mapsto u\}\sigma_L(T')\sigma_L(S')\sigma_L(A')$.

The property that $\mathcal{G}'$ is a rigid generalization of $\mathcal{G}'$ and $\mathcal{G}_1$ is obviously maintained during this transformation.

In Merge we have $A = A' = \emptyset$, $S = \{x_2 : \tilde{\nu} \triangleq \tilde{\nu}\} \cup S'$, $T = T'$, $\mathcal{G} = \{\omega_1 \triangleq \chi_1, \omega_2 \triangleq \chi_2\} \cup \mathcal{G}''$, and $\mathcal{G}' = \mathcal{G}''\{\omega_2 \mapsto \omega_1\} \cup \{\omega_1 \triangleq \chi_1\}$, where $\{\chi_1 : \tilde{\nu} \triangleq \tilde{\nu}\} \in S'$. We get $\mathcal{G}\sigma_L(T)\sigma_L(S)\sigma_L(\emptyset) = \{(\omega_1 \triangleq \chi_1, \omega_2 \triangleq \chi_2) \cup \mathcal{G}''\}\sigma_L(T')\sigma_L(\{\chi_2 : \tilde{\nu} \triangleq \tilde{\nu}\} \cup S') = \mathcal{G}_1$ and by uniqueness of $\chi_2$ follows $\mathcal{G}_1 = \{(\omega_1 \triangleq \chi_1, \omega_2 \triangleq \chi_2) \cup \mathcal{G}''\} \chi_2 \mapsto \tilde{\nu}\sigma_L(T')\sigma_L(S')$. Since $\sigma_L(S')$ also contains the mapping $\{\chi_1 \mapsto \tilde{\nu}\}$ we get $\mathcal{G}_1 = \{(\omega_1 \triangleq \chi_1, \omega_2 \triangleq \chi_2) \cup \mathcal{G}''\} \chi_2 \mapsto \chi_1\sigma_L(T')\sigma_L(S') = \mathcal{G}''\{\omega_2 \mapsto \omega_1\} \cup \{\omega_1 \triangleq \chi_1\}\sigma_L(T')\sigma_L(S')$.

The property that $\mathcal{G}'$ is a rigid generalization of $\mathcal{G}'$ and $\mathcal{G}_1$ is maintained because of the condition that from $\chi_1 \in \mathcal{V}_i$ follows $\chi_2 \notin \mathcal{V}_i$ forbids the instantiation of a term variable by a hedge variable.

\begin{theorem}[Soundness] If $\{x : \text{root}(\mathcal{G}_1) \triangleq \text{root}(\mathcal{G}_2)\}; \emptyset; \{x \doteq x\} \implies^* \emptyset; S; T; \mathcal{G}$ is a derivation in Gen(\mathcal{R}), then $\mathcal{G}$ is an $\mathcal{R}$-generalization of $\mathcal{G}_1$ and $\mathcal{G}_2$.
\end{theorem}

\begin{proof}
The assumptions of Lemma 2 hold for the initial configuration $\{x : \text{root}(\mathcal{G}_1) \triangleq \text{root}(\mathcal{G}_2)\}; \emptyset; \{x \doteq x\}$. Since Gen(\mathcal{R}) terminates on any input (Theorem 2), it follows that all the generalization variables from $S$ and $T$ are unique among all the other variables from $S$ and $T$. Moreover, $\mathcal{G}\sigma_L(T)\sigma_L(S) = \mathcal{G}_1$ and $\mathcal{G}\sigma_R(T)\sigma_R(S) = \mathcal{G}_2$. $\mathcal{G}$ is a rigid generalization of $\mathcal{G}_1$ and $\mathcal{G}_2$. To prove that $\mathcal{G}$ is an $\mathcal{R}$-generalization, it remains to show that the recursion from Definition 8 item 2 has been applied exhaustively. This follows from the fact that the store is complete, i.e., $\mathcal{G}\sigma_L(T)\sigma_L(S) = \mathcal{G}_1$ and $\mathcal{G}\sigma_R(T)\sigma_R(S) = \mathcal{G}_2$, and from the condition of the rule Solve that $\mathcal{R}(\text{top}(\tilde{v}, \mathcal{G}_1), \text{top}(\tilde{v}, \mathcal{G}_2))$ is either $\emptyset$ or $\{\}$. \hfill \qed
\end{proof}

\begin{corollary}[Soundness of the Store]
If $\{x : \text{root}(\mathcal{G}_1) \triangleq \text{root}(\mathcal{G}_2)\}; \emptyset; \{x \doteq x\} \implies^* \emptyset; S; T; \mathcal{G}$ is a derivation in Gen(\mathcal{R}), then $\mathcal{G}\sigma_L(T)\sigma_L(S) = \mathcal{G}_1$ and $\mathcal{G}\sigma_R(T)\sigma_R(S) = \mathcal{G}_2$.
\end{corollary}

Notice that Gen(\mathcal{R}) computes generalizations that do not have free term variables. Therefore, they are not considered in the completeness theorem. However, we show in section 6 that this restriction can be lifted by adding an additional transformation rule.

\begin{theorem}[Completeness] Let $\mathcal{G}$ be an $\mathcal{R}$-generalization of $\mathcal{G}_1$ and $\mathcal{G}_2$. Then Gen(\mathcal{R}) computes an $\mathcal{R}$-generalization $\mathcal{G}'$ of $\mathcal{G}_1$ and $\mathcal{G}_2$ such that $\mathcal{G} \leq \mathcal{G}'$.
\end{theorem}

\begin{proof}
By our assumption, $\mathcal{G}_1$ and $\mathcal{G}_2$ do not contain free variables. If $\mathcal{G}$ has a form $\{\text{root}(\mathcal{G}) \doteq x\}$, then $x$ must be a fresh variable and any generalization computed by Gen(\mathcal{R}) satisfies the theorem. Now assume root(\mathcal{G}) \doteq f(\tilde{v}) \in \mathcal{G}. Then we should have root(\mathcal{G}_1) \doteq f(\tilde{\chi}) \in \mathcal{G}_1$ and root(\mathcal{G}_2) \doteq f(\tilde{v}) \in \mathcal{G}_2$ and we can start the derivation with Step. We can make the next step immediately by Dec-S rule, taking the same alignment (from $\mathcal{R}(\text{top}(\tilde{\chi}, \mathcal{G}_1), \text{top}(\tilde{v}, \mathcal{G}_2))$ which is used in $f(\tilde{v})$ (since $\mathcal{G}$ is an $\mathcal{R}$-generalization, such an alignment exists). Further, if Merge is applicable, we make this step as long as possible.

\end{proof}
Theorem 5. For two bisimilar term-graphs, \( Gen(R) \) computes their join in the lattice generated by functional bisimulation.

Proof. It is easy to see that our algorithm returns only one answer for bisimilar graphs (since there is no branching at \( Dec-S \) rule) and the computed generalization contains no new free variables (the store is empty). Then the set \( T \) gives exactly a bisimulation, which justifies bisimilarity between the original term-graphs: \( R_T = \{(v,v) \mid \chi : v \triangleq v \in T \text{ for some } \chi \} \). The computed generalization \( G \) is the same as the term-graph \( G_{\mathcal{R}_T}^A \) associated to \( R_T \). (The node \( \chi \in G \) can be seen as the node \( (v,v) \in G_{\mathcal{R}_T}^A \) for each \( \chi : v \triangleq v \in T \).) By construction of \( T \), for each \( (v,v) \in R_T \), the access paths are not disjoint: \( acc(v) \cap acc(v) \neq \emptyset \) (otherwise there would be a new free variable in the generalization introduced by \( Dec-S \)). By Proposition 3.13 in [4], it implies that \( R_T \) is a minimal bisimulation. Therefore, from the constructive proof of Theorem 3.19 in [4] we conclude that \( G_{\mathcal{R}_T}^A \) (i.e. \( G \)) is the join of \( G_1 \) and \( G_2 \).

Example 5.3. Let \( G_1 = \{x_0 \triangleright f(x_1), x_1 \triangleright f(x_2), x_2 \triangleright f(x_3), x_3 \triangleright f(x_4), x_4 \triangleright f(x_5), x_5 \triangleright f(x_3)\} \) and \( G_2 = \{y_0 \triangleright f(y_1), y_1 \triangleright f(y_2), y_2 \triangleright f(y_3), y_3 \triangleright f(y_4), y_4 \triangleright f(y_5), y_5 \triangleright f(y_6), y_6 \triangleright f(y_7), y_7 \triangleright f(y_2)\} \). They are bisimilar. The algorithm computes their lgg \( \mathcal{G} = \{z_0 \triangleright f(z_1), z_1 \triangleright f(z_2), z_2 \triangleright f(z_3), z_3 \triangleright f(z_4), z_4 \triangleright f(z_5), z_5 \triangleright f(z_6), z_6 \triangleright f(z_7), z_7 \triangleright f(z_8), z_8 \triangleright f(z_3)\} \). It is the join in the lattice of the bisimilarity class of \( G_1 \) and \( G_2 \) [4].

6 Extensions

The algorithm \( Gen \) computes generalizations in which the new free variables are only hedge variables. One can relax this restriction a bit, permitting new free term variables in the
generalizations. We would only require that if two neighboring nodes in the generalization are labeled by free generalization variables, then both of them should be a term variable. It is not difficult to extend the algorithm Gen to an algorithm which computes such generalizations: One can simply continue processing the store to see whether a hedge variable generalizes two term sequences of the same length, and split such an AUT into n new AUTs between the elements of the sequences generalized by a new term variable, applying the following rule:

Split: Splitting a Node in the Store
\[ \emptyset; \{ x : (t_1, \ldots, t_n) \equiv (r_1, \ldots, r_n) \} \cup S; \ T; \{ Y \equiv X \} \cup \mathcal{G} \Rightarrow \]
\[ \emptyset; S \cup \{ x_i : t_i \equiv r_i \mid 1 \leq i \leq n \}; \ T; \mathcal{G}\{ Y \mapsto (y_1, \ldots, y_n) \} \cup \{ y_1 \equiv x_1, \ldots, y_n \equiv x_n \}, \]
where \( x_i \) are new free term variables and \( y_i \) are new recursion terms variables for all \( 1 \leq i \leq n \).

Another extension concerns permitting commutative function symbols. It is quite handy, since it allows to model the situations when the order of edges coming out of certain edges does not matter. We will treat the arguments of commutative functions not as sequences, but as multisets. Double curly brackets are used to denote multisets. The modification of Step rule for commutative symbols will have the following form:

Step-C: Simplification step for Commutative symbols
\[ \{ x : y \equiv z \} \cup A; \ S; \ T; \mathcal{G} \Rightarrow A_0 \cup A; \ S; \ T \cup \{ u : y \equiv z \}; \mathcal{G}\{ x \mapsto u \} \cup \{ u \equiv t \}, \]
where \( y \mapsto l(v_1, \ldots, v_n) \in \mathcal{G}_1, z \mapsto l(v_1, \ldots, v_m) \in \mathcal{G}_2, l \) is commutative, \([1, 1] \in \mathcal{R}(\mathcal{G}_1), \mathcal{G}_2)\), \( T \) does not contain an AUT of the form \( \ldots y \equiv z \), and \( u \) is a fresh recursion term variable. If \( n = m = 0 \) then \( t = l \) and \( A_0 = \emptyset \), otherwise \( t = l(X) \) and \( A_0 = \{ X : \{ v_1, \ldots, v_n \} \equiv \{ u_1, \ldots, u_m \} \} \) where \( X \) is a fresh hedge variable.

We follow our principle not to allow neighboring new hedge variables in generalizations. Therefore, the counterpart of Dec-S rule for multisets will have the form:

Dec-S-C: Decomposition and Solving for Commutative Symbols
\[ \{ X : \{ v_1, \ldots, v_n \} \equiv \{ u_1, \ldots, u_m \} \} \cup A; \ S; \ T; \mathcal{G} \Rightarrow \]
\[ A \cup \{ y_r : v_r \equiv u_r \mid 1 \leq r \leq k \}; \]
\[ S \cup \{ Y : \{ v_1, \ldots, v_n \} \backslash \{ v_{i_1}, \ldots, v_{i_k} \} \equiv \{ u_1, \ldots, u_m \} \backslash \{ u_{j_1}, \ldots, u_{j_k} \} \}; \]
\[ T; \mathcal{G}\sigma \cup \{ Z \equiv Y \}, \]
if \( \mathcal{R}_C(\mathcal{G}_1, \mathcal{G}_2) \) contains a sequence \( l_{[i_1, j_1] \cdots [i_k, j_k]}, k > 0 \). The \( y \)'s are fresh term variables, \( Y \) is a fresh hedge variables, \( Z \) is a fresh recursion hedge variable, and \( \sigma = \{ X \mapsto (y_1, \ldots, y_k, Z) \} \). If the new AUT has the form \( Y : \{ \} \equiv \{ \} \), then it is not added to \( S \) and \( Z \) does not appear in \( \sigma \).

The counterpart of Merge rule works with two multiset AUTs and checks whether the corresponding sides are equal as multisets, furthermore, we also merge sequences with multisets. This leads to the following two merge rules:

Merge-C1: Merging Nodes in the Store
\[ \emptyset; \{ \chi_1 : \{ v_1, \ldots, v_n \} \equiv \{ u_1, \ldots, u_m \} \} \cup \mathcal{V}_1 \cup \mathcal{V}_2; \ T; \{ \omega_1 \equiv \chi_1, \omega_2 \equiv \chi_2 \} \cup \mathcal{G} \Rightarrow \]
\[ \emptyset; S \cup \{ \chi_1 : \{ v_1, \ldots, v_n \} \equiv \{ u_1, \ldots, u_m \} \}; \ T; \mathcal{G}\{ \omega_2 \mapsto \omega_1 \} \cup \{ \omega_1 \equiv \chi_1 \}, \]
where \( \chi_1, \chi_2 \in \mathcal{V}_1 \cup \mathcal{V}_2 \) such that if \( \chi_1 \in \mathcal{V}_1 \), then \( \chi_1 \in \mathcal{V}_1 \).
Merge-C2: Merging Nodes in the Store

\[ \emptyset; \{ \chi_1 : (v_1, \ldots, v_n) \triangleq (u_1, \ldots, u_m) \}; \chi_2 : \{ \{v_1, \ldots, v_n\} \triangleq \{u_1, \ldots, u_m\}\} \cup S; \]

\[ T; \{ \omega_1 \rightarrow \chi_1, \omega_2 \rightarrow \chi_2 \} \cup G \implies \emptyset; \]

\[ S \cup \{ \chi_1 : (v_1, \ldots, v_n) \triangleq (u_1, \ldots, u_m)\}; \]

\[ T; \{ \omega_1 \rightarrow \chi_1 \} \cup \{ \omega_2 \rightarrow \chi_2 \}, \]

where \( \chi_1, \chi_2 \in \mathcal{V}_t \cup \mathcal{V}_s \) such that if \( \chi_1 \in \mathcal{V}_s \), then \( \chi_2 \notin \mathcal{V}_t \).

Notice that Merge-C2 implicitly arranges the elements in the multisets as needed, since the rules are applied exhaustively in all possible ways.

Termination, soundness and completeness of the algorithms for these extensions can be shown similarly to the corresponding properties for Gen(\( \mathcal{R} \)).

7 Conclusion

We have presented an anti-unification algorithm for (unranked) term-graphs, which are given as systems of recursion equations. The algorithm is sound, complete, and terminating, and uses a parameter, called rigidity function. The function selects common edges outgoing from the pair of nodes to be generalized. While longest common subsequence is the most intuitive instance of the rigidity function, the properties of the algorithm hold for any concrete rigid instance of the parameter. A refinement and a commutative extension of the algorithm are also presented. As a future work, extending simply typed lambda term anti-unification [11] to cyclic lambda terms [5] would provide a generalization of our results from a first-order language to a higher-order one.

References


