Term-Graph Anti-Unification

Alexander Baumgartner
Department of Computer Science (DCC), University of Chile, Santiago, Chile
abaumgar@dcc.uchile.cl

Temur Kutsia
Research Institute for Symbolic Computation, Johannes Kepler University Linz, Austria
kutsia@risc.jku.at

Jordi Levy
Artificial Intelligence Research Institute (IIIA), Spanish National Research Council, (CSIC), Barcelona, Spain
levy@iiia.csic.es

Mateu Villaret
Departament d'Informàtica, Matemàtica Aplicada i Estadística, Universitat de Girona, Girona, Spain
mateu.villaret@udg.edu

Abstract

We study anti-unification for possibly cyclic, unranked term-graphs and develop an algorithm, which computes a minimal complete set of generalizations for them. For bisimilar graphs the algorithm computes the join in the lattice generated by a functional bisimulation. 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.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases Cyclic term-graps, anti-unification, least general generalization

Digital Object Identifier 10.4230/LIPIcs.FSCD.2018.9

Funding Supported by the Austrian Science Fund (FWF) under the projects P 28789-N32 and J 3909-N31, by MINECO/FEDER projects TIN2015-71799-C2-1-P (RASO) and TIN2015-66293-R (LoCos), and by UdG project MPCUdG2016/055.

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., [4, 6, 15, 19, 20, 27]. Term-graphs can be represented in various ways, for instance, as constraints [6], hypergraphs [27], systems of recursion equations [4], or arrows in a category [15]. 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, μ-terms, control flow graphs, abstract semantic graphs, program dependency graphs, certain kinds of flowcharts, process graphs, etc.

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 9; pp. 9:1–9:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we study the anti-unification problem for term-graphs: Given two such graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ (maybe with cycles), our goal is to find a graph $\mathcal{G}$, which is a least general common generalization of $\mathcal{G}_1$ and $\mathcal{G}_2$. It means, there should exist variable substitutions σ_1 and σ_2 such that the instances of $\mathcal{G}$ with respect to them, i.e., the graphs $\mathcal{G}\sigma_1$ and $\mathcal{G}\sigma_2$, are equivalent to $\mathcal{G}_1$ and $\mathcal{G}_2$, 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 [22] 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.

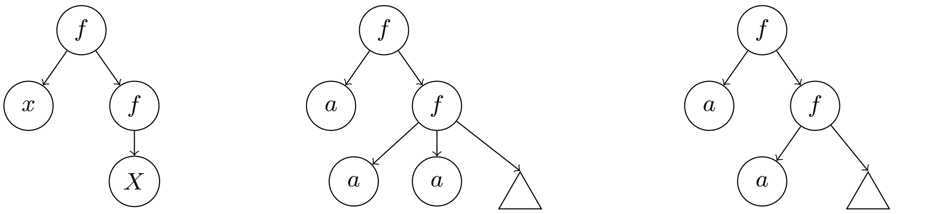
The intuition behind lggs is that they should contain "maximal similarities" between the input graphs and should abstract differences between them by variables uniformly. While this might sound similar to the problem of computing maximal common subgraphs (mcs) between graphs [23, 24], lggs, in general, 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, 22] to unranked cyclic graphs. In particular, we extend rigid anti-unification from terms to graphs. The rigid version is a more efficient variant 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. Properties of the latter are proved for arbitrary values of this parameter. As special cases of our results, we obtain anti-unification for ranked term-graphs, rational trees, μ-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 [26]. 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, 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, 13, 18, 21, 25]. A particularly interesting motivation comes from software code clone detection, where anti-unification has been successfully incorporated at the level of abstract syntax trees [14, 16, 28]. 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. Conclusions and the future work are discussed in Sect. 6.

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



Finite term
$f(x, f(X))$

Infinite non-rational term
$f(a, f(a, a, f(a, a, a, f(\ldots))))$

Infinite rational term
$f(a, f(a, f(a, f(\ldots)))$

Figure 1 Unranked terms and their tree representations.

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 ϵ.

▶ **Definition 1.** Given pairwise disjoint sets of unranked function symbols $\mathcal{F}$ (symbols without fixed arity), term variables $\mathcal{V}_\mathsf{t}$, and hedge variables $\mathcal{V}_\mathsf{s}$, an *unranked term* is a partial mapping $t : \mathbb{N}^* \to \mathcal{F} \cup \mathcal{V}_\mathsf{t} \cup \mathcal{V}_\mathsf{s}$ such that

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

A term t is *finite* if $dom(t)$ 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 $\mathcal{F}$, $\mathcal{V}_\mathsf{t}$, and $\mathcal{V}_\mathsf{s}$ is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V}_\mathsf{t}, \mathcal{V}_\mathsf{s})$ (respectively, $\mathcal{H}(\mathcal{F}, \mathcal{V}_\mathsf{t}, \mathcal{V}_\mathsf{s})$). 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, χ, ν, υ and ω 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 ϵ. Given a sequence $\tilde{s}$, the ith 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.** 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:

For a term t, we denote by $\mathcal{V}_\mathsf{t}(t)$, $\mathcal{V}_\mathsf{s}(t)$, and $\mathcal{V}(t)$ 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 3.** A *system of recursion equations* over $\mathcal{F}$, $\mathcal{V}_\mathsf{t}$, and $\mathcal{V}_\mathsf{s}$ is a set of equations $\{x_1 \doteq t_1, \ldots, x_n \doteq t_n, X_1 \doteq \tilde{s}_1, \ldots, X_m \doteq \tilde{s}_m\}$, where for all i, j, $1 \leq i < j \leq n$, $x_i \neq x_j$, all t's are finite terms, for all i, j, $1 \leq i < j \leq m$, $X_i \neq X_j$, and $\tilde{s}$'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 Γ, denoted by $root(\Gamma)$. It is always a term variable.

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

A system Γ is called *horizontally bounded* if no hedge variable is reachable from itself in Γ, i.e., Γ contains no horizontal cycle.[1] For instance, $\{\boldsymbol{x} \doteq f(\boldsymbol{x}), \boldsymbol{X} \doteq (\boldsymbol{x}, Y)\}$ is a horizontally bounded system, while $\{\boldsymbol{x} \doteq f(\boldsymbol{x}), \boldsymbol{X} \doteq (\boldsymbol{x}, \boldsymbol{X})\}$ 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: $\boldsymbol{x} \doteq f(\boldsymbol{\chi}_1, \dots, \boldsymbol{\chi}_n)$, $\boldsymbol{x} \doteq u$ where u is a free or bound term variable, and $\boldsymbol{X} \doteq (\upsilon_1, \dots, \upsilon_n)$ where $n \geq 0$ and each υ_i is a free or bound term or hedge variable.

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

- $\boldsymbol{x} \doteq f(\boldsymbol{\chi}_1, \dots, \boldsymbol{\chi}_n)$, where the $\boldsymbol{\chi}$'s are (not necessarily distinct) recursion variables, or
- $\boldsymbol{x} \doteq y$, where y is a free variable, or
- $\boldsymbol{X} \doteq Y$, where Y is a free variable.

For instance, $\{\boldsymbol{x} \doteq f(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{X}), \boldsymbol{y} \doteq g(\boldsymbol{x}), \boldsymbol{X} \doteq Y\}$ and $\{\boldsymbol{x} \doteq f(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{z}), \boldsymbol{y} \doteq g(\boldsymbol{x}), \boldsymbol{z} \doteq a\}$ are in canonical form, while $\{\boldsymbol{x} \doteq f(g(\boldsymbol{x}), \boldsymbol{X}), \boldsymbol{X} \doteq Y\}$, $\{\boldsymbol{x} \doteq f(\boldsymbol{y}, X), \boldsymbol{y} \doteq g(\boldsymbol{x})\}$, $\{\boldsymbol{x} \doteq f(\boldsymbol{y}, \boldsymbol{X}), \boldsymbol{y} \doteq g(\boldsymbol{x}), \boldsymbol{X} \doteq a\}$, and $\{\boldsymbol{x} \doteq f(\boldsymbol{y}, \boldsymbol{X}), \boldsymbol{y} \doteq g(\boldsymbol{x}), \boldsymbol{X} \doteq \boldsymbol{Y}, \boldsymbol{Y} \doteq Z\}$ 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 $\boldsymbol{x} \doteq \boldsymbol{y}$ and replace all occurrences of $\boldsymbol{x}$ by $\boldsymbol{y}$. If $\boldsymbol{y} = \boldsymbol{x}$, then replace the equation by $\boldsymbol{x} \doteq \bullet$, where $\bullet$ is some predefined constant from $\mathcal{F}$.
- Replace equations of the form $\boldsymbol{X} \doteq (\upsilon_1, \dots, \upsilon_n)$, $n > 1$, where υ's are free or bound term or hedge variables, by n new equations $\boldsymbol{Y}_i \doteq \upsilon_i$, $1 \leq i \leq n$, where $\boldsymbol{Y}_i$'s are fresh hedge variables, and replace each occurrence of $\boldsymbol{X}$ by $(\boldsymbol{Y}_1, \dots, \boldsymbol{Y}_n)$.
- Replace equations of the form $\boldsymbol{X} \doteq u$ by $\boldsymbol{x} \doteq u$ and replace each occurrence of $\boldsymbol{X}$ by $\boldsymbol{x}$, where u is a free or bound term variable and $\boldsymbol{x}$ is a fresh term variable.
- Remove trivial equations of the form $\boldsymbol{X} \doteq \boldsymbol{Y}$ and replace all occurrences of $\boldsymbol{X}$ by $\boldsymbol{Y}$.
- Remove equations of the form $\boldsymbol{X} \doteq \epsilon$ and remove each occurrence of $\boldsymbol{X}$.

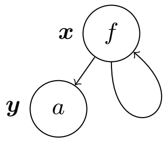
Essentially, this canonicalization extends the canonicalization from [4] by four steps dealing with hedge variables. These steps split each equation of the form $\boldsymbol{X} \doteq (s_1, \dots, s_n)$ into n new equations (one for each s_i), and, eventually only those are retained for which s_i is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

[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.

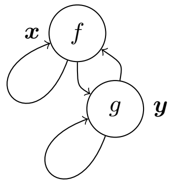
Intuitively, canonical systems of recursion equations can be naturally represented by graphs: The nodes will be the recursion variables; a node $\boldsymbol{x}$ will be connected to a node $\boldsymbol{\chi}$ by an edge if the system contains an equation $\boldsymbol{x} \doteq f(\dots,\boldsymbol{\chi},\dots)$;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

▶ **Definition 4.** A *term-graph* is a system of recursion equations in canonical form.

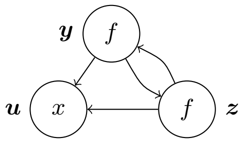
▶ **Example 5.** We show some term-graphs and their defining recursion equations.

1. Graph: 

Equations: $\boldsymbol{x} \doteq f(\boldsymbol{y},\boldsymbol{x})$
$\boldsymbol{y} \doteq a$

2. Graph: 

Equations: $\boldsymbol{x} \doteq f(\boldsymbol{x},\boldsymbol{y})$
$\boldsymbol{y} \doteq g(\boldsymbol{y},\boldsymbol{x})$

3. Graph: 

Equations: $\boldsymbol{y} \doteq f(\boldsymbol{u},\boldsymbol{z})$
$\boldsymbol{z} \doteq f(\boldsymbol{u},\boldsymbol{y})$
$\boldsymbol{u} \doteq x$

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 $\mathcal{G}$ will be used to denote term-graphs.

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

The set of nodes of a term-graph $\mathcal{G}$ is denoted by $\mathit{nodes}(\mathcal{G})$. If $\boldsymbol{x} \in \mathit{nodes}(\mathcal{G})$ and $\boldsymbol{\nu}$ is its ith successor (i.e., $\boldsymbol{x} \doteq t \in \mathcal{G}$ for some t and $\boldsymbol{\nu}$ is ith argument of t), we will write $\boldsymbol{x} \to_i \boldsymbol{\nu}$. An *access path* of $\boldsymbol{\nu} \in \mathit{nodes}(\mathcal{G})$ is a possibly empty finite sequence of positive natural numbers $(i_1,\dots,i_j)$ such that there exist $\boldsymbol{\chi}_1,\dots,\boldsymbol{\chi}_{j-1} \in \mathit{nodes}(\mathcal{G})$ with $\mathit{root}(\mathcal{G}) \to_{i_1} \boldsymbol{\chi}_1 \to_{i_2} \cdots \to_{i_{j-1}} \boldsymbol{\chi}_{j-1} \to_{i_j} \boldsymbol{\nu}$. A node may have several access paths. The set of all access paths of a node $\boldsymbol{\chi}$ is denoted by $\mathit{acc}(\boldsymbol{\chi})$.

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 $\tilde{\mathcal{G}}$ to denote them.

3 Bisimilarity Relation

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

▶ **Definition 6.** Let $\Gamma_1 = \{\boldsymbol{\chi}_1 \doteq s_1,\dots,\boldsymbol{\chi}_n \doteq s_n\}$ and $\Gamma_2 = \{\boldsymbol{\nu}_1 \doteq q_1,\dots,\boldsymbol{\nu}_m \doteq q_m\}$ be two systems of recursion equations. Then R is a *bisimulation* from Γ_1 to Γ_2 iff

- R is a binary relation with the domain $\{\boldsymbol{\chi}_1,\dots,\boldsymbol{\chi}_n\}$ and codomain $\{\boldsymbol{\nu}_1,\dots,\boldsymbol{\nu}_m\}$.
- The roots of Γ_1 and Γ_2 are related: $\boldsymbol{\chi}_1 \, R \, \boldsymbol{\nu}_1$.

- If $\chi_i \, R \, \nu_j$, $\chi_i \doteq l_1(\chi_1^i, \dots, \chi_{k_i}^i) \in \Gamma_1$, $k_i \geq 0$, and $\nu_j \doteq l_2(\nu_1^j, \dots, \nu_{k_j}^j) \in \Gamma_2$, $k_j \geq 0$, then $l_1 = l_2$, $k_i = k_j$, and $\chi_u \, R \, \nu_u$ 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 7.** 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 $\mathcal{G}_1 \gtrsim \mathcal{G}_2$ 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, $\triangle\mathcal{G} \gtrsim \mathcal{G} \gtrsim \nabla\mathcal{G}$.

▶ **Example 8.** Let $\mathcal{G}$ be the term graph $\{\boldsymbol{x} \doteq f(\boldsymbol{y}, \boldsymbol{z}), \boldsymbol{y} \doteq a, \boldsymbol{z} = f(\boldsymbol{y}, \boldsymbol{x})\}$. Then $\triangle\mathcal{G}$ is the infinite rational term depicted in Fig. 1 in Example 2, and $\nabla\mathcal{G}$ is the first graph in Example 5.

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) $\mathit{nodes}(\mathcal{G}_R^A) = R$, $\mathit{root}(\mathcal{G}_R^A) = (\mathit{root}(\mathcal{G}_1), \mathit{root}(\mathcal{G}_2))$, the label of each $(\chi_1, \chi_2) \in \mathit{nodes}(\mathcal{G}_R^A)$ is that of χ_1 (which is the same as the label of χ_2); (ii) if $\chi_1 \in \mathit{nodes}(\mathcal{G}_1)$, $\chi_2 \in \mathit{nodes}(\mathcal{G}_2)$, $(\chi_1, \chi_2) \in R$, $\chi_1 \to_i \chi_1'$, and $\chi_2 \to_i \chi_2'$, then in $\mathcal{G}_R^A$ we have $(\chi_1, \chi_2) \to_i (\chi_1', \chi_2')$.

4 Substitutions and Generalizations

The notions related to substitutions, formulated for finite unranked terms and hedges in [22], 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, e.g., $\{x \mapsto f(a, f(a, \dots)), X \mapsto \epsilon, Y \mapsto (X, g(Y, g(Y, \dots, Y), Y))\}$, which stands for the substitution that maps every variable to itself except x, X, and Y that are mapped respectively to $f(a, f(a, \dots))$, ϵ, and $(X, g(Y, g(Y, \dots, Y), Y))$.

The lower case Greek letters are used to denote substitutions, with the exception of the identity substitution for which we write *Id*. The *domain* and *range* of a substitution σ are defined in the usual way: $\mathit{dom}(\sigma) = \{\chi \in \mathcal{V} \mid \sigma(\chi) \neq \chi\}$ and $\mathit{ran}(\sigma) = \{\sigma(\chi) \mid \chi \in \mathit{dom}(\sigma)\}$.

Substitutions can be *applied* to terms and hedges using the congruences $\sigma(f(s_1, \dots, s_n)) = f(\sigma(s_1), \dots, \sigma(s_n))$ and $\sigma(s_1, \dots, s_n) = (\sigma(s_1), \dots, \sigma(s_n))$. 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 $\tilde{q} \preceq \tilde{s}$, then we write $\tilde{s} \simeq \tilde{q}$. If $\tilde{s} \preceq \tilde{q}$ and $\tilde{s} \not\simeq \tilde{q}$, then we say that $\tilde{s}$ is *strictly more general* than $\tilde{q}$ and write $\tilde{s} \prec \tilde{q}$.

The *composition* of two substitutions σ and ϑ, 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 σ_1 is *more general* than σ_2 with respect to a set of variables $\mathcal{X} \subseteq \mathcal{V}$, written $\sigma_1 \preceq_{\mathcal{X}} \sigma_2$, if there exists ϑ such that $\chi\sigma_1\vartheta = \chi\sigma_2$, for each $\chi \in \mathcal{X}$. The relations $\simeq$ and $\prec$ are extended to substitutions: $\sigma_1 \simeq_{\mathcal{X}} \sigma_2$ means $\sigma_1 \preceq_{\mathcal{X}} \sigma_2$ and $\sigma_2 \preceq_{\mathcal{X}} \sigma_1$, and $\sigma_1 \prec_{\mathcal{X}} \sigma_2$ means $\sigma_1 \preceq_{\mathcal{X}} \sigma_2$ and $\sigma_1 \not\simeq_{\mathcal{X}} \sigma_2$.

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. Hedge variables in the image remain unchanged. For instance, assume the term-graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are given by the systems of recursion equations: $\mathcal{G}_1 = \{\boldsymbol{x} \doteq f(\boldsymbol{y}, \boldsymbol{x}), \boldsymbol{y} \doteq a\}$, and $\mathcal{G}_2 = \{\boldsymbol{x} \doteq g(\boldsymbol{X}, \boldsymbol{x}, \boldsymbol{X}), \boldsymbol{X} \doteq Y\}$. 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 $\{x \mapsto \mathit{root}(\mathcal{G}_1), X \mapsto \epsilon, Y \mapsto (X, \mathit{root}(\mathcal{G}_2))\}$. The bound variables in $\mathcal{G}_1$ and $\mathcal{G}_2$ should be appropriately renamed to guarantee that the names are distinct from each other and from free variables.

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 σ be a substitution and $\mathcal{G}$ be a term-graph. Then the term-graph $\sigma(\mathcal{G})$, the instance of $\mathcal{G}$ under σ, is obtained by canonicalizing the following flat horizontally bounded system of recursion equations: $\{\chi \doteq \sigma(\tilde{s}) \mid \chi \doteq \tilde{s} \in \mathcal{G}\} \cup \mathcal{G}_1 \cup \cdots \cup \mathcal{G}_n$, where $\mathcal{G}_1, \ldots, \mathcal{G}_n$ are all term-graphs whose roots appear in $\mathit{ran}(\sigma)$. Substitution application naturally extends to term-graph hedges.

▶ **Example 9.** Let $\mathcal{G}$ be the term-graph:

$$\mathcal{G} = \{\boldsymbol{x}_0 \doteq f(\boldsymbol{x}_0, \boldsymbol{X}_1, \boldsymbol{x}_1, \boldsymbol{X}_1, \boldsymbol{x}_2),\ \boldsymbol{X}_1 \doteq X,\ \boldsymbol{x}_1 \doteq g(\boldsymbol{X}_2, \boldsymbol{x}_2, \boldsymbol{X}_2),\ \boldsymbol{X}_2 \doteq Y,\ \boldsymbol{x}_2 \doteq x\}.$$

Let $\sigma = \{x \mapsto \mathit{root}(\mathcal{G}_1), X \mapsto (\mathit{root}(\mathcal{G}_2), X), Y \mapsto \epsilon\}$, $\mathcal{G}_1 = \{\boldsymbol{y} \doteq f(\boldsymbol{y})\}$, $\mathcal{G}_2 = \{\boldsymbol{z} \doteq a\}$. Then

$$\sigma(\mathcal{G}) = \{\boldsymbol{x}_0 \doteq f(\boldsymbol{x}_0, \boldsymbol{z}, \boldsymbol{Z}, \boldsymbol{x}_1, \boldsymbol{z}, \boldsymbol{Z}, \boldsymbol{y}),\ \boldsymbol{z} \doteq a,\ \boldsymbol{Z} \doteq X,\ \boldsymbol{x}_1 \doteq g(\boldsymbol{y}),\ \boldsymbol{y} \doteq f(\boldsymbol{y})\}.$$

The notion of *more general* term-graphs and term-graph hedges is defined modulo bisimilarity: $\tilde{\mathcal{G}}_1$ is more general than $\tilde{\mathcal{G}}_2$, if there is a substitution σ such that $\tilde{\mathcal{G}}_1\sigma \sim \tilde{\mathcal{G}}_2$. We reuse the symbol $\preceq$ for this relation over term-graphs and term-graph hedges, and also write $\tilde{\mathcal{G}}_1 \simeq \tilde{\mathcal{G}}_2$ if $\tilde{\mathcal{G}}_1 \preceq \tilde{\mathcal{G}}_2$ and $\tilde{\mathcal{G}}_2 \preceq \tilde{\mathcal{G}}_1$. For the strict part of $\preceq$ we reuse $\prec$. Analogously for substitutions: A substitution over term-graphs σ_1 is *more general* than a substitution over term-graphs σ_2 with respect to a set of variables $\mathcal{X} \subseteq \mathcal{V}$, if there exists ϑ such that $\chi\sigma_1\vartheta \sim \chi\sigma_2$ for each $\chi \in \mathcal{X}$. Also in this case we reuse the $\preceq$ symbol and write $\sigma_1 \preceq_{\mathcal{X}} \sigma_2$ (and similarly for the relations $\simeq$ and $\prec$ for substitutions).

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

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

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

Soundness: Each $\tilde{\mathcal{G}} \in G$ is a generalization of both $\tilde{\mathcal{G}}_1$ and $\tilde{\mathcal{G}}_2$.
Completeness: For each generalization $\tilde{\mathcal{G}}'$ of $\tilde{\mathcal{G}}_1$ and $\tilde{\mathcal{G}}_2$, there exists $\tilde{\mathcal{G}} \in G$ such that $\tilde{\mathcal{G}}' \preceq \tilde{\mathcal{G}}$.

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

Minimality: For each $\tilde{\mathcal{G}}'_1, \tilde{\mathcal{G}}'_2 \in G$, if $\tilde{\mathcal{G}}'_1 \preceq \tilde{\mathcal{G}}'_2$ then $\tilde{\mathcal{G}}'_1 = \tilde{\mathcal{G}}'_2$.

▶ **Lemma 10.** *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 [22]. ◀

▶ **Theorem 11.** *For any term-graph hedges $\tilde{\mathcal{G}}_1$ and $\tilde{\mathcal{G}}_2$ there exists their minimal complete set of generalizations. This set is finite and unique modulo $\simeq$ and $\sim$.*

Proof. Note that $\mathcal{G} \sim \triangle\mathcal{G}$ for all $\mathcal{G}$. Let $\tilde{\mathcal{G}}_1 = (\mathcal{G}^1_1, \dots, \mathcal{G}^1_n)$ and $\tilde{\mathcal{G}}_2 = (\mathcal{G}^2_1, \dots, \mathcal{G}^2_m)$. By Lemma 10, the hedges $(\triangle\mathcal{G}^1_1, \dots, \triangle\mathcal{G}^1_n)$ and $(\triangle\mathcal{G}^2_1, \dots, \triangle\mathcal{G}^2_m)$ have a finite minimal complete set of generalizations, unique modulo $\simeq$. ◀

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 [22].[2]

▶ **Definition 12** (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.

For instance, if $\mathcal{R}$ computes the set of all longest common subsequences, then $\mathcal{R}(abcda, bcad) = \{b[2,1]c[3,2]a[5,3],\ b[2,1]c[3,2]d[4,4]\}$.

The *top symbol* of a term is defined as $top(x) = x$ for any variable x, and $top(f(\tilde{s})) = f$ for any term $f(\tilde{s})$. The notion is extended to hedges: $top(X) = X$ and $top(s_1, \dots, s_n) = (top(s_1), \dots, top(s_n))$. If $\{\boldsymbol{\chi}_1 \doteq s_1, \dots, \boldsymbol{\chi}_1 \doteq s_n\} \subseteq \mathcal{G}$, $n > 0$, then we define $top(\boldsymbol{\chi}_1, \dots, \boldsymbol{\chi}_n, \mathcal{G})$ as $top(s_1, \dots, s_n)$. Moreover, we define $top(\mathcal{G}) = top(root(\mathcal{G}), \mathcal{G})$.

▶ **Definition 13** ($\mathcal{R}$-Generalization)**.** Given two term-graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ (without common free and bound variables) and the rigidity function $\mathcal{R}$, we say that a term-graph $\mathcal{G}$ that generalizes both $\mathcal{G}_1$ and $\mathcal{G}_2$ is their *generalization with respect to $\mathcal{R}$*, or, shortly, an $\mathcal{R}$-generalization, if either

- $\mathcal{R}(top(\mathcal{G}_1), top(\mathcal{G}_2)) \in \{\emptyset, \{\epsilon\}\}$ and $\mathcal{G} = \{\boldsymbol{x} \doteq y\}$, where $\boldsymbol{x}$ is a new bound term variable and y is a new free term variable, or
- $f[1,1] \in \mathcal{R}(top(\mathcal{G}_1), top(\mathcal{G}_2))$ for some f and $\mathcal{G} = \{root(\mathcal{G}) \doteq f(\tilde{\boldsymbol{\chi}})\} \cup \mathcal{Y} \cup \mathcal{G}'_1 \cup \cdots \cup \mathcal{G}'_n$, where $\tilde{\boldsymbol{\chi}}$ does not contain pairs of consecutive hedge recursion variables.
  The sequence $\tilde{\boldsymbol{\chi}}$, the set $\mathcal{Y}$, and the graphs $\mathcal{G}'_1, \dots, \mathcal{G}'_n$ are defined as follows:
  For $i = 1, 2$, the original graph $\mathcal{G}_i$ contains an equation $root(\mathcal{G}_i) \doteq f(\tilde{\mathbf{v}}_i)$ and there exists an alignment $g_1[i_1, j_1] \cdots g_n[i_n, j_n] \in \mathcal{R}(top(\tilde{\mathbf{v}}_1, \mathcal{G}_1), top(\tilde{\mathbf{v}}_2, \mathcal{G}_2))$, satisfying the following conditions:
  1. If we remove all hedge recursion variables that occur as elements of $\tilde{\boldsymbol{\chi}}$, we get a sequence of term recursion variables $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$, such that $\boldsymbol{x}_k = root(\mathcal{G}'_k)$ and each $\mathcal{G}'_k$ contains an equation of the form $\boldsymbol{x}_k \doteq g_k(\tilde{\mathbf{v}}_k)$ for all $1 \leq k \leq n$, and

[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].

2. For every $1 \le k \le n$, there exists a pair of term recursion variables $\boldsymbol{y}_k^1$ and $\boldsymbol{y}_k^2$ such that $\tilde{\mathbf{v}}_1|_{i_k} = \boldsymbol{y}_k^1$, $\tilde{\mathbf{v}}_2|_{j_k} = \boldsymbol{y}_k^2$, and $\mathcal{G}'_k$ is an $\mathcal{R}$-generalization of $subgraph(\mathcal{G}_1, \boldsymbol{y}_k^1)$ and $subgraph(\mathcal{G}_2, \boldsymbol{y}_k^2)$.
3. $\mathcal{Y} = \{\boldsymbol{Y}_1 \doteq Z_1, \dots, \boldsymbol{Y}_m \doteq Z_m\}$, where $\boldsymbol{Y}_1, \dots, \boldsymbol{Y}_m$ are all hedge recursion variables in $\tilde{\boldsymbol{\chi}}$ and $Z_1, \dots, Z_m$ are new free hedge variables.

▶ **Example 14.** Let $\mathcal{R}$ compute the set of all longest common subsequences and let $\mathcal{G}_1 = \{\boldsymbol{x}_0 \doteq f(\boldsymbol{x}_1, \boldsymbol{x}_2),\ \boldsymbol{x}_1 \doteq g(\boldsymbol{x}_2, \boldsymbol{x}_2),\ \boldsymbol{x}_2 \doteq a\}$ and $\mathcal{G}_2 = \{\boldsymbol{y}_0 \doteq f(\boldsymbol{y}_1, \boldsymbol{y}_0, \boldsymbol{y}_2, \boldsymbol{y}_0),\ \boldsymbol{y}_1 \doteq g,\ \boldsymbol{y}_2 \doteq a\}$. The term graph $\{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{Z}_1, \boldsymbol{z}_2, \boldsymbol{Z}_1),\ \boldsymbol{z}_1 \doteq g(\boldsymbol{Z}_2),\ \boldsymbol{z}_2 \doteq a,\ \boldsymbol{Z}_1 \doteq Z_1,\ \boldsymbol{Z}_2 \doteq Z_2\}$ is an $\mathcal{R}$-generalization of $\mathcal{G}_1$ and $\mathcal{G}_2$ while $\{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{Z}_1, \boldsymbol{z}_2, \boldsymbol{Z}_1),\ \boldsymbol{z}_1 \doteq g(\boldsymbol{Z}_2, \boldsymbol{Z}_2),\ \boldsymbol{z}_2 \doteq a,\ \boldsymbol{Z}_1 \doteq Z_1,\ \boldsymbol{Z}_2 \doteq Z_2\}$ and $\{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{Z}_1, \boldsymbol{z}_2, \boldsymbol{Z}_1),\ \boldsymbol{z}_1 \doteq z,\ \boldsymbol{z}_2 \doteq a,\ \boldsymbol{Z}_1 \doteq Z_1\}$ are not.

5 The Algorithm

We present our anti-unification algorithm as a rule-based algorithm that works on quadruples $A; S; T; \mathcal{G}$, called configurations. Here A, S, and T are sets of anti-unification triples and $\mathcal{G}$ 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**

$\{x : \boldsymbol{y} \triangleq \boldsymbol{z}\} \uplus A;\ S;\ T;\ \mathcal{G} \Longrightarrow A_0 \cup A;\ S;\ T \cup \{\boldsymbol{u} : \boldsymbol{y} \triangleq \boldsymbol{z}\};\ \mathcal{G}\{x \mapsto \boldsymbol{u}\} \cup \{\boldsymbol{u} \doteq t\}$,

where $\boldsymbol{y} \doteq l(\tilde{\mathbf{v}}) \in \mathcal{G}_1$, $\boldsymbol{z} \doteq l(\tilde{\mathfrak{v}}) \in \mathcal{G}_2$, $l[1,1] \in \mathcal{R}(top(\boldsymbol{y}, \mathcal{G}_1), top(\boldsymbol{z}, \mathcal{G}_2))$, T does not contain an AUT of the form $\_ : \boldsymbol{y} \triangleq \boldsymbol{z}$, and $\boldsymbol{u}$ is a fresh recursion term variable. If $\tilde{\mathbf{v}} = \tilde{\mathfrak{v}} = \epsilon$ then $t = l$ and $A_0 = \emptyset$, otherwise $t = l(X)$ and $A_0 = \{X : \tilde{\mathbf{v}} \triangleq \tilde{\mathfrak{v}}\}$ where X is a fresh hedge variable.

Dec-S: **Decomposition and Solving**

$\{X : \tilde{\mathbf{v}} \triangleq \tilde{\mathfrak{v}}\} \uplus A;\ S;\ T;\ \mathcal{G} \Longrightarrow$
$\quad A \cup \{y_k : \tilde{\mathbf{v}}|_{i_k} \triangleq \tilde{\mathfrak{v}}|_{j_k} \mid 1 \le k \le n\}$;
$\quad S \cup \{Y_0 : \tilde{\mathbf{v}}|_0^{i_1} \triangleq \tilde{\mathfrak{v}}|_0^{j_1}\} \cup \{Y_k : \tilde{\mathbf{v}}|_{i_k}^{i_{k+1}} \triangleq \tilde{\mathfrak{v}}|_{j_k}^{j_{k+1}} \mid 1 \le k \le n-1\} \cup \{Y_n : \tilde{\mathbf{v}}|_{i_n}^{|\tilde{\mathbf{v}}|+1} \triangleq \tilde{\mathfrak{v}}|_{j_n}^{|\tilde{\mathfrak{v}}|+1}\}$;
$\quad T;\ \mathcal{G}\sigma \cup \{\boldsymbol{Z}_0 \doteq Y_0, \dots, \boldsymbol{Z}_n \doteq Y_n\}$,

if $\mathcal{R}(top(\tilde{\mathbf{v}}, \mathcal{G}_1), top(\tilde{\mathfrak{v}}, \mathcal{G}_2))$ 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 $\boldsymbol{Z}$'s are fresh recursion hedge variables, and the substitution is $\sigma = \{X \mapsto (\boldsymbol{Z}_0, y_1, \boldsymbol{Z}_1, \dots, \boldsymbol{Z}_{n-1}, y_n, \boldsymbol{Z}_n)\}$. For each $1 \le i \le n$, if the new AUT has the form $Y_i : \epsilon \triangleq \epsilon$, then it is not added to S and $\boldsymbol{Z}_i$ does not appear in σ.

Solve: **Solving**

$\{\chi : \tilde{\boldsymbol{\nu}} \triangleq \tilde{\boldsymbol{\upsilon}}\} \cup A;\ S;\ T;\ \mathcal{G} \Longrightarrow A;\ S \cup \{\chi : \tilde{\boldsymbol{\nu}} \triangleq \tilde{\boldsymbol{\upsilon}}\};\ T;\ \mathcal{G}\{\chi \mapsto \boldsymbol{\omega}\} \cup \{\boldsymbol{\omega} \doteq \chi\}$,

if $\mathcal{R}(top(\tilde{\boldsymbol{\nu}}, \mathcal{G}_1), top(\tilde{\boldsymbol{\upsilon}}, \mathcal{G}_2)) = \emptyset$ or $\mathcal{R}(top(\tilde{\boldsymbol{\nu}}, \mathcal{G}_1), top(\tilde{\boldsymbol{\upsilon}}, \mathcal{G}_2)) = \{\epsilon\}$. The variable $\boldsymbol{\omega}$ is a fresh recursion variable. If $\chi \in \mathcal{V}_\mathsf{t}$, then $\boldsymbol{\omega} \in \mathcal{V}_\mathsf{t}$ and if $\chi \in \mathcal{V}_\mathsf{s}$, then $\boldsymbol{\omega} \in \mathcal{V}_\mathsf{s}$.

Share: **Sharing**

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

Merge: **Merging Nodes in the Store**

$$\emptyset;\ \{\chi_1 : \tilde{\boldsymbol{\nu}} \triangleq \tilde{\boldsymbol{\upsilon}},\ \chi_2 : \tilde{\boldsymbol{\nu}} \triangleq \tilde{\boldsymbol{\upsilon}}\} \cup S;\ \ T;\ \{\boldsymbol{\omega}_1 \doteq \chi_1, \boldsymbol{\omega}_2 \doteq \chi_2\} \cup \mathcal{G} \Longrightarrow$$
$$\emptyset;\ S \cup \{\chi_1 : \tilde{\boldsymbol{\nu}} \triangleq \tilde{\boldsymbol{\upsilon}}\};\ T;\ \mathcal{G}\{\boldsymbol{\omega}_2 \mapsto \boldsymbol{\omega}_1\} \cup \{\boldsymbol{\omega}_1 \doteq \chi_1\},$$

where $\chi_1, \chi_2 \in \mathcal{V}_\mathsf{t} \cup \mathcal{V}_\mathsf{s}$ such that if $\chi_1 \in \mathcal{V}_\mathsf{s}$, then $\chi_2 \notin \mathcal{V}_\mathsf{t}$.

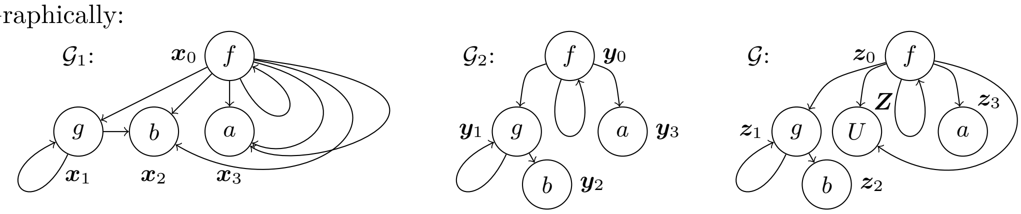
The rules never generate the AUTs of the form $X : \epsilon \triangleq \epsilon$. To compute $\mathcal{R}$-generalizations of $\mathcal{G}_1$ and $\mathcal{G}_2$, we start with $\{x : root(\mathcal{G}_1) \triangleq root(\mathcal{G}_2)\}, \emptyset, \emptyset, \{\boldsymbol{x} \doteq x\}$ and apply the rules on the selected AUTs in all possible ways. The obtained procedure is denoted by $Gen(\mathcal{R})$.

The notation $\Longrightarrow^*$ 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., $A; S : T; \mathcal{G} \Longrightarrow_{\mathsf{Step}} A'; S' : T'; \mathcal{G}'$ for the transformation with the rule Simplification Step.

► **Example 15.** Let $\mathcal{R}$ be the longest common subsequence. Then the term-graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ below have a unique $\mathcal{R}$-lgg $\mathcal{G}$:

$$\mathcal{G}_1 = \{\boldsymbol{x}_0 \doteq f(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_0, \boldsymbol{x}_3, \boldsymbol{x}_2, \boldsymbol{x}_3),\ \boldsymbol{x}_1 \doteq g(\boldsymbol{x}_1, \boldsymbol{x}_2),\ \boldsymbol{x}_2 \doteq b,\ \boldsymbol{x}_3 \doteq a\}.$$
$$\mathcal{G}_2 = \{\boldsymbol{y}_0 \doteq f(\boldsymbol{y}_1, \boldsymbol{y}_0, \boldsymbol{y}_3),\ \boldsymbol{y}_1 \doteq g(\boldsymbol{y}_1, \boldsymbol{y}_2),\ \boldsymbol{y}_2 \doteq b,\ \boldsymbol{y}_3 \doteq a\}.$$
$$\mathcal{G} = \{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{Z}_1, \boldsymbol{z}_0, \boldsymbol{z}_3, \boldsymbol{Z}_1),\ \boldsymbol{z}_1 \doteq g(\boldsymbol{z}_1, \boldsymbol{z}_2),\ \boldsymbol{Z} \doteq U,\ \boldsymbol{z}_3 \doteq a,\ \boldsymbol{z}_2 \doteq b\}.$$

Graphically:



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

$\{u_0 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0\}; \emptyset; \emptyset; \{\boldsymbol{z}_0 \doteq u_0\} \Longrightarrow_{\mathsf{Step}}$

$\{U_0 : (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_0, \boldsymbol{x}_3, \boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq (\boldsymbol{y}_1, \boldsymbol{y}_0, \boldsymbol{y}_3)\}; \emptyset; \{\boldsymbol{z}_0 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0\};$

$\quad \{\boldsymbol{z}_0 \doteq f(\boldsymbol{Z}_0),\ \boldsymbol{Z}_0 \doteq U_0\} \Longrightarrow_{\mathsf{Dec\text{-}S}}$

(Choosing the common subsequence: $g[1,1]f[4,2]a[5,3]$), corresponding to the node pairs $\boldsymbol{x}_1$ and $\boldsymbol{y}_1$, $\boldsymbol{x}_0$ and $\boldsymbol{y}_0$, the second occurrence of $\boldsymbol{x}_3$ and $\boldsymbol{y}_3$.)

$\{u_1 : \boldsymbol{x}_1 \triangleq \boldsymbol{y}_1,\ u_2 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0,\ u_3 : \boldsymbol{x}_3 \triangleq \boldsymbol{y}_3\};\ \{U_1 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon,\ U_2 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon\};$

$\quad \{\boldsymbol{z}_0 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0\};$

$\quad \{\boldsymbol{z}_0 \doteq f(\boldsymbol{u}_1, \boldsymbol{Z}_1, \boldsymbol{u}_2, \boldsymbol{u}_3, \boldsymbol{Z}_2),$

$\quad\quad \boldsymbol{u}_1 \doteq u_1,\ \boldsymbol{Z}_1 \doteq U_1,\ \boldsymbol{u}_2 \doteq u_2,\ \boldsymbol{u}_3 \doteq u_3,\ \boldsymbol{Z}_2 \doteq U_2\} \Longrightarrow_{\mathsf{Step}}$

$\{u_2 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0,\ u_3 : \boldsymbol{x}_3 \triangleq \boldsymbol{y}_3,\ U_3 : (\boldsymbol{x}_1, \boldsymbol{x}_2) \triangleq (\boldsymbol{y}_1, \boldsymbol{y}_2)\};$

$\quad \{U_1 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon,\ U_2 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon\};\ \{\boldsymbol{z}_0 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0,\ \boldsymbol{z}_1 : \boldsymbol{x}_1 \triangleq \boldsymbol{y}_1\};$

$$\{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{Z}_1, \boldsymbol{u}_2, \boldsymbol{u}_3, \boldsymbol{Z}_2),$$
$$\boldsymbol{z}_1 \doteq g(\boldsymbol{Z}_3),\ \boldsymbol{Z}_1 \doteq U_1,\ \boldsymbol{u}_2 \doteq u_2,\ \boldsymbol{u}_3 \doteq u_3,\ \boldsymbol{Z}_2 \doteq U_2,\ \boldsymbol{Z}_3 \doteq U_3\} \Longrightarrow_{\mathsf{Share}}$$
$$\{u_3 : \boldsymbol{x}_3 \triangleq \boldsymbol{y}_3,\ U_3 : (\boldsymbol{x}_1, \boldsymbol{x}_2) \triangleq (\boldsymbol{y}_1, \boldsymbol{y}_2)\};$$
$$\{U_1 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon,\ U_2 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon\};\ \{\boldsymbol{z}_0 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0,\ \boldsymbol{z}_1 : \boldsymbol{x}_1 \triangleq \boldsymbol{y}_1\};$$
$$\{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{Z}_1, \boldsymbol{z}_0, \boldsymbol{u}_3, \boldsymbol{Z}_2),$$
$$\boldsymbol{z}_1 \doteq g(\boldsymbol{Z}_3),\ \boldsymbol{Z}_1 \doteq U_1,\ \boldsymbol{u}_3 \doteq u_3,\ \boldsymbol{Z}_2 \doteq U_2,\ \boldsymbol{Z}_3 \doteq U_3\} \Longrightarrow_{\mathsf{Step}}$$
$$\{U_3 : (\boldsymbol{x}_1, \boldsymbol{x}_2) \triangleq (\boldsymbol{y}_1, \boldsymbol{y}_2)\};$$
$$\{U_1 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon,\ U_2 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon\};\ \{\boldsymbol{z}_0 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0,\ \boldsymbol{z}_1 : \boldsymbol{x}_1 \triangleq \boldsymbol{y}_1,\ \boldsymbol{z}_3 : \boldsymbol{x}_3 \triangleq \boldsymbol{y}_3\};$$
$$\{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{Z}_1, \boldsymbol{z}_0, \boldsymbol{z}_3, \boldsymbol{Z}_2),$$
$$\boldsymbol{z}_1 \doteq g(\boldsymbol{Z}_3),\ \boldsymbol{Z}_1 \doteq U_1,\ \boldsymbol{z}_3 \doteq a,\ \boldsymbol{Z}_2 \doteq U_2,\ \boldsymbol{Z}_3 \doteq U_3\} \Longrightarrow_{\mathsf{Dec\text{-}S}}$$

(Choosing the common subsequence: $g[1,1]b[2,2]$), corresponding to the node pairs $\boldsymbol{x}_1$ and $\boldsymbol{y}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{y}_2$.)

$$\{u_4 : \boldsymbol{x}_1 \triangleq \boldsymbol{y}_1,\ u_5 : \boldsymbol{x}_2 \triangleq \boldsymbol{y}_2\};$$
$$\{U_1 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon,\ U_2 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon\};\ \{\boldsymbol{z}_0 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0,\ \boldsymbol{z}_1 : \boldsymbol{x}_1 \triangleq \boldsymbol{y}_1,\ \boldsymbol{z}_3 : \boldsymbol{x}_3 \triangleq \boldsymbol{y}_3\};$$
$$\{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{Z}_1, \boldsymbol{z}_0, \boldsymbol{z}_3, \boldsymbol{Z}_2),$$
$$\boldsymbol{z}_1 \doteq g(\boldsymbol{u}_4, \boldsymbol{u}_5),\ \boldsymbol{Z}_1 \doteq U_1,\ \boldsymbol{z}_3 \doteq a,\ \boldsymbol{Z}_2 \doteq U_2,\ \boldsymbol{u}_4 \doteq u_4,\ \boldsymbol{u}_5 \doteq u_5\} \Longrightarrow_{\mathsf{Share}}$$
$$\{u_5 : \boldsymbol{x}_2 \triangleq \boldsymbol{y}_2\};$$
$$\{U_1 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon,\ U_2 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon\};\ \{\boldsymbol{z}_0 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0,\ \boldsymbol{z}_1 : \boldsymbol{x}_1 \triangleq \boldsymbol{y}_1,\ \boldsymbol{z}_3 : \boldsymbol{x}_3 \triangleq \boldsymbol{y}_3\};$$
$$\{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{Z}_1, \boldsymbol{z}_0, \boldsymbol{z}_3, \boldsymbol{Z}_2),$$
$$\boldsymbol{z}_1 \doteq g(\boldsymbol{z}_1, \boldsymbol{u}_5),\ \boldsymbol{Z}_1 \doteq U_1,\ \boldsymbol{z}_3 \doteq a,\ \boldsymbol{Z}_2 \doteq U_2,\ \boldsymbol{u}_5 \doteq u_5\} \Longrightarrow_{\mathsf{Step}}$$
$$\emptyset;\ \{U_1 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon,\ U_2 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon\};$$
$$\{\boldsymbol{z}_0 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0,\ \boldsymbol{z}_1 : \boldsymbol{x}_1 \triangleq \boldsymbol{y}_1,\ \boldsymbol{z}_3 : \boldsymbol{x}_3 \triangleq \boldsymbol{y}_3,\ \boldsymbol{z}_2 : \boldsymbol{x}_2 \triangleq \boldsymbol{y}_2\};$$
$$\{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{Z}_1, \boldsymbol{z}_0, \boldsymbol{z}_3, \boldsymbol{Z}_2),$$
$$\boldsymbol{z}_1 \doteq g(\boldsymbol{z}_1, \boldsymbol{z}_2),\ \boldsymbol{Z}_1 \doteq U_1,\ \boldsymbol{z}_3 \doteq a,\ \boldsymbol{Z}_2 \doteq U_2,\ \boldsymbol{z}_2 \doteq b\} \Longrightarrow_{\mathsf{Merge}}$$
$$\emptyset;\ \{U_1 : (\boldsymbol{x}_2, \boldsymbol{x}_3) \triangleq \epsilon\};\ \{\boldsymbol{z}_0 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0,\ \boldsymbol{z}_1 : \boldsymbol{x}_1 \triangleq \boldsymbol{y}_1,\ \boldsymbol{z}_3 : \boldsymbol{x}_3 \triangleq \boldsymbol{y}_3,\ \boldsymbol{z}_2 : \boldsymbol{x}_2 \triangleq \boldsymbol{y}_2\};$$
$$\{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{Z}_1, \boldsymbol{z}_0, \boldsymbol{z}_3, \boldsymbol{Z}_1),\ \boldsymbol{z}_1 \doteq g(\boldsymbol{z}_1, \boldsymbol{z}_2),\ \ \boldsymbol{Z}_1 \doteq U_1,\ \boldsymbol{z}_3 \doteq a,\ \boldsymbol{z}_2 \doteq b\}.$$

The obtained generalization is equal to $\mathcal{G}$ modulo renaming variables. The store and the trail suggest how to obtain the original term-graphs from the computed generalization. For instance, to obtain $\mathcal{G}_1$ from $\mathcal{G}$, we just apply the substitution $\{U_1 \mapsto (\boldsymbol{x}_2, \boldsymbol{x}_3)\}$ to $\mathcal{G}$. In the obtained term-graph we will have $\boldsymbol{x}_2 \doteq b$ and $\boldsymbol{x}_3 \doteq a$ alongside to $\boldsymbol{z}_2 \doteq b$ and $\boldsymbol{z}_3 \doteq a$, but it will be bisimilar to $\mathcal{G}_1$.

▶ **Example 16.** Let $\mathcal{G}_1 = \{\boldsymbol{x}_0 \doteq f(\boldsymbol{x}_1, \boldsymbol{x}_2), \boldsymbol{x}_1 \doteq g(\boldsymbol{x}_0, \boldsymbol{x}_3), \boldsymbol{x}_2 \doteq a, \boldsymbol{x}_3 \doteq b\}$ and $\mathcal{G}_2 = \{\boldsymbol{y}_0 \doteq f(\boldsymbol{y}_1, \boldsymbol{y}_2), \boldsymbol{y}_1 \doteq h(\boldsymbol{y}_0, \boldsymbol{y}_3), \boldsymbol{y}_2 \doteq a, \boldsymbol{y}_3 \doteq b\}$. Then the algorithm ends with $\emptyset, \{z : \boldsymbol{x}_1 \triangleq \boldsymbol{y}_1\}$, $\{\boldsymbol{z}_0 : \boldsymbol{x}_0 \triangleq \boldsymbol{y}_0, \boldsymbol{z}_2 : \boldsymbol{x}_2 \triangleq \boldsymbol{y}_2\}, \{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{z}_2), \boldsymbol{z}_1 \doteq z, \boldsymbol{z}_2 \doteq a\}$. Obtaining $\mathcal{G}_1$ from the computed generalization can be illustrated as $\{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1, \boldsymbol{z}_2), \boldsymbol{z}_1 \doteq z, \boldsymbol{z}_2 \doteq a\}\{z \mapsto \boldsymbol{x}_1\} = \{\boldsymbol{z}_0 \doteq f(\boldsymbol{x}_1, \boldsymbol{z}_2), \boldsymbol{z}_2 \doteq a, \boldsymbol{x}_1 \doteq g(\boldsymbol{x}_0, \boldsymbol{x}_3), \boldsymbol{x}_0 \doteq f(\boldsymbol{x}_1, \boldsymbol{x}_2), \boldsymbol{x}_2 \doteq a, \boldsymbol{x}_3 \doteq b\} \sim \mathcal{G}_1$.

▶ **Theorem 17** (Termination)**.** *The procedure $Gen(\mathcal{R})$ terminates on any input and produces a configuration $\emptyset$; S; T; $\mathcal{G}$, 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 $\boldsymbol{x} : t_1 \triangleq t_2$ be $size(t_1) + size(t_2) + 1$, and the size of $\boldsymbol{X} : \tilde{s}_1 \triangleq \tilde{s}_2$ 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 (χ_1, χ_2), where $\chi_1 \in \mathcal{G}_1$ and $\chi_2 \in \mathcal{G}_2$. Any other rule strictly decreases the size of A or, in case of Merge, the size of S. Moreover, Merge does not change the size of A. The rule Dec-S can introduce only finite branching. Therefore, the algorithm terminates. ◀

▶ **Definition 18.** Given a set A of AUTs where all the generalization variables are pairwise distinct. We define two substitutions that can be obtained from A:

$$\sigma_L(A) := \{\chi \mapsto \tilde{\mathbf{v}} \mid \chi : \tilde{\mathbf{v}} \triangleq \tilde{\mathbf{\upsilon}} \in A\} \qquad \sigma_R(A) := \{\chi \mapsto \tilde{\mathbf{\upsilon}} \mid \chi : \tilde{\mathbf{v}} \triangleq \tilde{\mathbf{\upsilon}} \in A\}$$

▶ **Lemma 19** (Transformation Invariant). *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 $\mathcal{G}\sigma_L(T)\sigma_L(S)\sigma_L(A) = \mathcal{G}_1$ 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 $A; S; T; \mathcal{G} \Longrightarrow A'; S'; T'; \mathcal{G}'$ 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) $\tilde{\mathbf{v}} = \tilde{\mathbf{\upsilon}} = \epsilon$, and (ii) $\tilde{\mathbf{v}} \neq \epsilon$ or $\tilde{\mathbf{\upsilon}} \neq \epsilon$. We only illustrate the more general case (ii) since the two proofs are largely identical. Therefore, we have $A = \{x : \boldsymbol{y} \triangleq \boldsymbol{z}\} \cup (A' \setminus \{X : \tilde{\mathbf{v}} \triangleq \tilde{\mathbf{\upsilon}}\})$, $S = S'$, $T \cup \{\boldsymbol{u} : \boldsymbol{y} \triangleq \boldsymbol{z}\} = T'$, and $\mathcal{G}\{x \mapsto \boldsymbol{u}\} \cup \{\boldsymbol{u} \doteq l(X)\} = \mathcal{G}'$, where $\boldsymbol{y} \doteq l(\tilde{\mathbf{v}}) \in \mathcal{G}_1$, $\boldsymbol{z} \doteq l(\tilde{\mathbf{\upsilon}}) \in \mathcal{G}_2$ and $\boldsymbol{u}$, X are fresh. Since $\boldsymbol{u}$, X are fresh, all the generalization variables from A', S', T' are still unique among all the other variables from A', S', T'. Obviously, $\mathcal{G}\sigma_L(T)\sigma_L(S)\sigma_L(A) = \mathcal{G}\sigma_L(T)\sigma_L(S')\sigma_L(\{x : \boldsymbol{y} \triangleq \boldsymbol{z}\} \cup (A' \setminus \{X : \tilde{\mathbf{v}} \triangleq \tilde{\mathbf{\upsilon}}\})) = \mathcal{G}_1$. From the uniqueness of x, and by definition of substitution application follows that $\mathcal{G}_1 = \mathcal{G}\{x \mapsto \boldsymbol{y}\}\sigma_L(T)\sigma_L(S')\sigma_L(A' \setminus \{X : \tilde{\mathbf{v}} \triangleq \tilde{\mathbf{\upsilon}}\}) = (\mathcal{G}\{x \mapsto \boldsymbol{y}\} \cup \{\boldsymbol{y} \doteq l(\tilde{\mathbf{v}})\})\sigma_L(T)\sigma_L(S')\sigma_L(A' \setminus \{X : \tilde{\mathbf{v}} \triangleq \tilde{\mathbf{\upsilon}}\})$. From the uniqueness of X follows $\mathcal{G}_1 = (\mathcal{G}\{x \mapsto \boldsymbol{y}\} \cup \{\boldsymbol{y} \doteq l(X)\})\sigma_L(T)\sigma_L(S')\sigma_L(A')$. Finally, from the uniqueness of $\boldsymbol{u}$ follows $\mathcal{G}_1 = (\mathcal{G}\{x \mapsto \boldsymbol{y}\} \cup \{\boldsymbol{y} \doteq l(X)\})\{\boldsymbol{y} \mapsto \boldsymbol{u}\}\sigma_L(T \cup \{\boldsymbol{u} : \boldsymbol{y} \triangleq \boldsymbol{z}\})\sigma_L(S')\sigma_L(A') = \mathcal{G}'\sigma_L(T')\sigma_L(S')\sigma_L(A')$.

Since Step can't lead to consecutive hedge variables and $l[1, 1] \in \mathcal{R}(top(\boldsymbol{y}, \mathcal{G}_1), top(\boldsymbol{z}, \mathcal{G}_2))$, 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 $A = \{X : \tilde{\mathbf{v}} \triangleq \tilde{\mathbf{\upsilon}}\} \cup (A' \setminus \{y_k : \tilde{\mathbf{v}}|_{i_k} \triangleq \tilde{\mathbf{\upsilon}}|_{j_k} \mid 1 \leq k \leq n\})$, $S \cup \{Y_0 : \tilde{\mathbf{v}}|_0^{i_1} \triangleq \tilde{\mathbf{\upsilon}}|_0^{j_1}\} \cup \{Y_k : \tilde{\mathbf{v}}|_{i_k}^{i_{k+1}} \triangleq \tilde{\mathbf{\upsilon}}|_{j_k}^{j_{k+1}} \mid 1 \leq k \leq n-1\} \cup \{Y_n : \tilde{\mathbf{v}}|_{i_n}^{|\tilde{\mathbf{v}}|+1} \triangleq \tilde{\mathbf{\upsilon}}|_{j_n}^{|\tilde{\mathbf{\upsilon}}|+1}\} = S'$, $T = T'$, and $\mathcal{G}\sigma \cup \{\boldsymbol{Z}_0 \doteq Y_0, \ldots, \boldsymbol{Z}_n \doteq Y_n\} = \mathcal{G}'$, where $\mathcal{R}(top(\tilde{\mathbf{v}}, \mathcal{G}_1), top(\tilde{\mathbf{\upsilon}}, \mathcal{G}_2))$ contains a sequence $l_1[i_1, j_1] \cdots l_n[i_n, j_n]$, $n > 0$, the y's, Y's, and $\boldsymbol{Z}$'s are fresh, and $\sigma = \{X \mapsto (\boldsymbol{Z}_0, y_1, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{n-1}, y_n, \boldsymbol{Z}_n)\}$. Since all the variables introduced by the transformation are fresh, all the generalization variables from A', S', T' are still unique. We get $\mathcal{G}\sigma_L(T)\sigma_L(S)\sigma_L(A) = \mathcal{G}\sigma_L(T')\sigma_L(S' \setminus (\{Y_0 : \tilde{\mathbf{v}}|_0^{i_1} \triangleq \tilde{\mathbf{\upsilon}}|_0^{j_1}\} \cup \{Y_k : \tilde{\mathbf{v}}|_{i_k}^{i_{k+1}} \triangleq \tilde{\mathbf{\upsilon}}|_{j_k}^{j_{k+1}} \mid 1 \leq k \leq n-1\} \cup \{Y_n : \tilde{\mathbf{v}}|_{i_n}^{|\tilde{\mathbf{v}}|+1} \triangleq \tilde{\mathbf{\upsilon}}|_{j_n}^{|\tilde{\mathbf{\upsilon}}|+1}\}))\sigma_L(\{X : \tilde{\mathbf{v}} \triangleq \tilde{\mathbf{\upsilon}}\} \cup (A' \setminus \{y_k : \tilde{\mathbf{v}}|_{i_k} \triangleq \tilde{\mathbf{\upsilon}}|_{j_k} \mid 1 \leq k \leq n\})) = \mathcal{G}_1$. By uniqueness of X, follows $\mathcal{G}_1 = \mathcal{G}\{X \mapsto \tilde{\mathbf{v}}\}\sigma_L(T')\sigma_L(S' \setminus (\{Y_0 : \tilde{\mathbf{v}}|_0^{i_1} \triangleq \tilde{\mathbf{\upsilon}}|_0^{j_1}\} \cup \{Y_k : \tilde{\mathbf{v}}|_{i_k}^{i_{k+1}} \triangleq \tilde{\mathbf{\upsilon}}|_{j_k}^{j_{k+1}} \mid 1 \leq

$k \leq n-1\} \cup \{Y_n : \tilde{\mathbf{v}}|_{i_n}^{|\tilde{\mathbf{v}}|+1} \triangleq \tilde{\mathbf{\upsilon}}|_{j_n}^{|\tilde{\mathbf{\upsilon}}|+1}\}))\sigma_L(A' \setminus \{y_k : \tilde{\mathbf{v}}|_{i_k} \triangleq \tilde{\mathbf{\upsilon}}|_{j_k} \mid 1 \leq k \leq n\})$. Now observe that $\mathcal{G}\{X \mapsto \tilde{\mathbf{v}}\}$ is equivalent to $\mathcal{G}\{X \mapsto (Y_0, y_1, Y_1, \ldots, Y_{n-1}, y_n, Y_n)\}\{Y_0 \mapsto \tilde{\mathbf{v}}|_0^{i_1}\}\{Y_k \mapsto \tilde{\mathbf{v}}|_{i_k}^{i_{k+1}} \mid 1 \leq k \leq n-1\}\{Y_n \mapsto \tilde{\mathbf{v}}|_{i_n}^{|\tilde{\mathbf{v}}|+1}\}\{y_k \mapsto \tilde{\mathbf{v}}|_{i_k} \mid 1 \leq k \leq n\}$, therefore $\mathcal{G}_1 = \mathcal{G}\{X \mapsto (Y_0, y_1, Y_1, \ldots, Y_{n-1}, y_n, Y_n)\}\{y_k \mapsto \tilde{\mathbf{v}}|_{i_k} \mid 1 \leq k \leq n\}\sigma_L(T')\sigma_L(S')\sigma_L(A' \setminus \{y_k : \tilde{\mathbf{v}}|_{i_k} \triangleq \tilde{\mathbf{\upsilon}}|_{j_k} \mid 1 \leq k \leq n\}) = \mathcal{G}\{X \mapsto (Y_0, y_1, Y_1, \ldots, Y_{n-1}, y_n, Y_n)\}\sigma_L(T')\sigma_L(S')\sigma_L(A') = (\mathcal{G}\{X \mapsto (\boldsymbol{Z}_0, y_1, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{n-1}, y_n, \boldsymbol{Z}_n)\} \cup \{\boldsymbol{Z}_0 \doteq Y_0, \ldots, \boldsymbol{Z}_n \doteq Y_n\})\sigma_L(T')\sigma_L(S')\sigma_L(A') = \mathcal{G}'\sigma_L(T')\sigma_L(S')\sigma_L(A')$.

Since Dec-S cannot lead to consecutive hedge variables, it follows that $\mathcal{G}'$ is a rigid generalization of $\mathcal{G}'$ and $\mathcal{G}_1$.

We omit the case of Solve because it is very similar to the case of Step.

In Share we have $A = \{x : \boldsymbol{y} \triangleq \boldsymbol{z}\} \uplus A'$, $S = S'$, $T = T'$, and $\mathcal{G}\{x \mapsto \boldsymbol{u}\} = \mathcal{G}'$, where $\{\boldsymbol{u} : \boldsymbol{y} \triangleq \boldsymbol{z}\} \in T$. Uniqueness of generalization variables from A', S', T' is obviously maintained. We get $\mathcal{G}\sigma_L(T)\sigma_L(S)\sigma_L(A) = \mathcal{G}\sigma_L(T')\sigma_L(S')\sigma_L(\{x : \boldsymbol{y} \triangleq \boldsymbol{z}\} \uplus A') = \mathcal{G}_1$ and by uniqueness of x follows $\mathcal{G}_1 = \mathcal{G}\{x \mapsto \boldsymbol{y}\}\sigma_L(T')\sigma_L(S')\sigma_L(A')$. The trail $\{\boldsymbol{u} : \boldsymbol{y} \triangleq \boldsymbol{z}\} \in T$ tells us that there is already a recursion variable $\boldsymbol{u}$ in $\mathcal{G}$ that represents the node $\boldsymbol{y}$ in $\mathcal{G}_1$. Therefore, instead of substituting x with $\boldsymbol{y}$ we may as well substitute it with $\boldsymbol{u}$. This consideration leads to $\mathcal{G}_1 = \mathcal{G}\{x \mapsto \boldsymbol{u}\}\sigma_L(T')\sigma_L(S')\sigma_L(A')$.

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 $A = A' = \emptyset$, $S = \{\chi_2 : \tilde{\mathbf{v}} \triangleq \tilde{\mathbf{\upsilon}}\} \uplus S'$, $T = T'$, $\mathcal{G} = \{\boldsymbol{\omega}_1 \doteq \chi_1, \boldsymbol{\omega}_2 \doteq \chi_2\} \uplus \mathcal{G}''$, and $\mathcal{G}' = \mathcal{G}''\{\boldsymbol{\omega}_2 \mapsto \boldsymbol{\omega}_1\} \cup \{\boldsymbol{\omega}_1 \doteq \chi_1\}$, where $\{\chi_1 : \tilde{\mathbf{v}} \triangleq \tilde{\mathbf{\upsilon}}\} \in S'$. We get $\mathcal{G}\sigma_L(T)\sigma_L(S)\sigma_L(\emptyset) = (\{\boldsymbol{\omega}_1 \doteq \chi_1, \boldsymbol{\omega}_2 \doteq \chi_2\} \uplus \mathcal{G}'')\sigma_L(T')\sigma_L(\{\chi_2 : \tilde{\mathbf{v}} \triangleq \tilde{\mathbf{\upsilon}}\} \uplus S') = \mathcal{G}_1$ and by uniqueness of χ_2 follows $\mathcal{G}_1 = (\{\boldsymbol{\omega}_1 \doteq \chi_1, \boldsymbol{\omega}_2 \doteq \chi_2\} \uplus \mathcal{G}'')\{\chi_2 \mapsto \tilde{\mathbf{v}}\}\sigma_L(T')\sigma_L(S')$. Since $\sigma_L(S')$ also contains the mapping $\{\chi_1 \mapsto \tilde{\mathbf{v}}\}$ we get $\mathcal{G}_1 = (\{\boldsymbol{\omega}_1 \doteq \chi_1, \boldsymbol{\omega}_2 \doteq \chi_2\} \uplus \mathcal{G}'')\{\chi_2 \mapsto \chi_1\}\sigma_L(T')\sigma_L(S') = (\mathcal{G}''\{\boldsymbol{\omega}_2 \mapsto \boldsymbol{\omega}_1\} \cup \{\boldsymbol{\omega}_1 \doteq \chi_1\})\sigma_L(T')\sigma_L(S')$.

The property that $\mathcal{G}'$ is a rigid generalization of $\mathcal{G}'$ and $\mathcal{G}_1$ is maintained because of the condition that from $\chi_1 \in \mathcal{V}_\mathsf{s}$ follows $\chi_2 \notin \mathcal{V}_\mathsf{t}$ forbids the instantiation of a term variable by a hedge variable. ◀

▶ **Theorem 20** (Soundness). *If $\{x : root(\mathcal{G}_1) \triangleq root(\mathcal{G}_2)\}; \emptyset; \emptyset; \{\boldsymbol{x} \doteq x\} \Longrightarrow^* \emptyset; S; T; \mathcal{G}$ is a derivation in $Gen(\mathcal{R})$, then $\mathcal{G}$ is an $\mathcal{R}$-generalization of $\mathcal{G}_1$ and $\mathcal{G}_2$.*

Proof. The assumptions of Lemma 19 hold for the initial configuration $\{x : root(\mathcal{G}_1) \triangleq root(\mathcal{G}_2)\}; \emptyset; \emptyset; \{\boldsymbol{x} \doteq x\}$. Since $Gen(\mathcal{R})$ terminates on any input (Theorem 17), 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 $\mathcal{G}\sigma_R(T)\sigma_R(S) = \mathcal{G}_2$, and $\mathcal{G}$ is a rigid generalization of $\mathcal{G}$ and $\mathcal{G}_i$ where $i \in \{1, 2\}$. Obviously $\mathcal{G}$ is a 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 13 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 $\mathcal{R}(top(\tilde{\mathbf{v}}, \mathcal{G}_1), top(\tilde{\mathbf{\upsilon}}, \mathcal{G}_2))$ is either $\emptyset$ or $\{\epsilon\}$. ◀

▶ **Corollary 21** (Soundness of the Store). *If $\{x : root(\mathcal{G}_1) \triangleq root(\mathcal{G}_2)\}; \emptyset; \emptyset; \{\boldsymbol{x} \doteq x\} \Longrightarrow^* \emptyset; S; T; \mathcal{G}$ is a derivation in $Gen(\mathcal{R})$, then $\mathcal{G}\sigma_L(T)\sigma_L(S) = \mathcal{G}_1$ and $\mathcal{G}\sigma_R(T)\sigma_R(S) = \mathcal{G}_2$.*

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 [12] that this restriction can be lifted by adding an additional transformation rule.

▶ **Theorem 22** (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 $\mathcal{G} \preceq \mathcal{G}'$.*

Proof. By our assumption, $\mathcal{G}_1$ and $\mathcal{G}_2$ do not contain free variables. If $\mathcal{G}$ has a form $\{root(\mathcal{G}) \doteq x\}$, then x must be a fresh variable and any generalization computed by $Gen(\mathcal{R})$ satisfies the theorem. Now assume $root(\mathcal{G}) \doteq f(\tilde{\boldsymbol{\upsilon}}) \in \mathcal{G}$. Then we should have $root(\mathcal{G}_1) \doteq f(\tilde{\boldsymbol{\chi}}) \in \mathcal{G}_1$ and $root(\mathcal{G}_2) \doteq f(\tilde{\boldsymbol{\nu}}) \in \mathcal{G}_2$ and we can start the derivation with Step. We can make the next step immediately by Dec-S rule, taking the same alignment (from $\mathcal{R}(top(\tilde{\boldsymbol{\chi}}, \mathcal{G}_1), top(\tilde{\boldsymbol{\nu}}, \mathcal{G}_2))$) which is used in $f(\tilde{\boldsymbol{\upsilon}})$ (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.

After these steps, in the set of new AUTs we will have only those which have counterparts for term variables occurring in $\tilde{\boldsymbol{\upsilon}}$. In the store there will be those AUTs which are generalized by hedge variables in $\tilde{\boldsymbol{\upsilon}}$. It can be that we merged more variables than it is done in $\tilde{\boldsymbol{\upsilon}}$, but it does not harm, since we are going to compute a generalization that is less general than $\mathcal{G}$. The trail will store the seen pair of the nodes (in this case the roots of $\mathcal{G}_1$ and $\mathcal{G}_2$). The generalization graph will contain the equation $root(\mathcal{G}') \doteq f(\tilde{\boldsymbol{\omega}})$, that corresponds to the root equation of $\mathcal{G}$, maybe with more shared variables. The bound term variables from $\tilde{\boldsymbol{\upsilon}}$ will have their counterparts in $\tilde{\boldsymbol{\omega}}$, but the equations which correspond to those variables in the current version of $\mathcal{G}'$ will have fresh free variables in the right hand side.

Next, we will pick an AUT in the new configuration. Its generalization variable has a unique counterpart in $\mathcal{G}$, which suggests how to make the next step, basically repeating the reasoning as above, unless the AUT has the form $\boldsymbol{z} : \boldsymbol{x} \triangleq \boldsymbol{y}$ and $\boldsymbol{z}' : \boldsymbol{x} \triangleq \boldsymbol{y}$ is already in the trail. We will use the Sharing rule to make the step. It can be horizontal or vertical sharing.

If it is a horizontal sharing, then it does not matter whether those nodes in $\mathcal{G}$ which correspond to $\boldsymbol{z}$ and $\boldsymbol{z}'$ are shared. If they are, then our construction of $\mathcal{G}'$ at this place directly imitates the structure of $\mathcal{G}$. If they are not, the $\mathcal{G}$ at this place is an expansion of $\mathcal{G}'$, but this operation preserves bisimilarity. In the vertical sharing, in addition to the above considered ones, it is also possible that at this place $\mathcal{G}$ is a collapsed version of $\mathcal{G}'$. But again, bisimilarity is preserved. Note that the construction of our derivation is not influenced by whether a particular node of $\mathcal{G}$ has already been seen or not. They are used to guide the construction, and the same node might guide more than one steps.

Iterating this process, eventually we stop with a generalization $\mathcal{G}'$ such that $\mathcal{G} \preceq \mathcal{G}'$. ◀

▶ **Theorem 23.** *For two bisimilar term-graphs, $Gen(\mathcal{R})$ 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: $R_T = \{(\boldsymbol{\nu}, \boldsymbol{\upsilon}) \mid \boldsymbol{\chi} : \boldsymbol{\nu} \triangleq \boldsymbol{\upsilon} \in T \text{ for some } \boldsymbol{\chi}\}$. The computed generalization $\mathcal{G}$ is the same as the term-graph $\mathcal{G}^A_{R_T}$ associated to R_T. (The node $\boldsymbol{\chi} \in \mathcal{G}$ can be seen as the node $(\boldsymbol{\nu}, \boldsymbol{\upsilon}) \in \mathcal{G}^A_{R_T}$ for each $\boldsymbol{\chi} : \boldsymbol{\nu} \triangleq \boldsymbol{\upsilon} \in T$.) By construction of T, for each $(\boldsymbol{\nu}, \boldsymbol{\upsilon}) \in R_T$, the access paths are not disjoint: $acc(\boldsymbol{\nu}) \cap acc(\boldsymbol{\upsilon}) \neq \emptyset$ (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 $\mathcal{G}^A_{R_T}$ (i.e. $\mathcal{G}$) is the join of $\mathcal{G}_1$ and $\mathcal{G}_2$. ◀

▶ **Example 24.** Let $\mathcal{G}_1 = \{\boldsymbol{x}_0 \doteq f(\boldsymbol{x}_1), \boldsymbol{x}_1 \doteq f(\boldsymbol{x}_2), \boldsymbol{x}_2 \doteq f(\boldsymbol{x}_3), \boldsymbol{x}_3 \doteq f(\boldsymbol{x}_4), \boldsymbol{x}_4 \doteq f(\boldsymbol{x}_5), \boldsymbol{x}_5 \doteq f(\boldsymbol{x}_3)\}$ and $\mathcal{G}_2 = \{\boldsymbol{y}_0 \doteq f(\boldsymbol{y}_1), \boldsymbol{y}_1 \doteq f(\boldsymbol{y}_2), \boldsymbol{y}_2 \doteq f(\boldsymbol{y}_3), \boldsymbol{y}_3 \doteq f(\boldsymbol{y}_4), \boldsymbol{y}_4 \doteq f(\boldsymbol{y}_5), \boldsymbol{y}_5 \doteq f(\boldsymbol{y}_6), \boldsymbol{y}_6 \doteq f(\boldsymbol{y}_7), \boldsymbol{y}_7 \doteq f(\boldsymbol{y}_2)\}$. They are bisimilar. The algorithm computes their lgg $\mathcal{G} = \{\boldsymbol{z}_0 \doteq f(\boldsymbol{z}_1), \boldsymbol{z}_1 \doteq f(\boldsymbol{z}_2), \boldsymbol{z}_2 \doteq f(\boldsymbol{z}_3), \boldsymbol{z}_3 \doteq f(\boldsymbol{z}_4), \boldsymbol{z}_4 \doteq f(\boldsymbol{z}_5), \boldsymbol{z}_5 \doteq f(\boldsymbol{z}_6), \boldsymbol{z}_6 \doteq f(\boldsymbol{z}_7), \boldsymbol{z}_7 \doteq f(\boldsymbol{z}_8), \boldsymbol{z}_8 \doteq f(\boldsymbol{z}_3)\}$. It is the join in the lattice of the bisimilarity class of $\mathcal{G}_1$ and $\mathcal{G}_2$ [4].

6 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. 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

1 Hassan Aït-Kaci and Gabriella Pasi. Lattice operations on terms over similar signatures. In *Pre-proceedings of the 27th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'17)*, 2017. URL: `https://arxiv.org/abs/1709.00964`.

2 María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. ACUOS: A system for modular ACU generalization with subtyping and inheritance. In Fermé and Leite [17], pages 573–581. `doi:10.1007/978-3-319-11558-0`.

3 María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. A modular order-sorted equational generalization algorithm. *Inf. Comput.*, 235:98–136, 2014. `doi:10.1016/j.ic.2014.01.006`.

4 Zena M. Ariola and Jan Willem Klop. Equational term graph rewriting. *Fundam. Inform.*, 26(3/4):207–240, 1996.

5 Zena M. Ariola and Jan Willem Klop. Lambda calculus with explicit recursion. *Inf. Comput.*, 139(2):154–233, 1997. `doi:10.1006/inco.1997.2651`.

6 Hendrik Pieter Barendregt, Marko C. J. D. van Eekelen, John R. W. Glauert, Richard Kennaway, Marinus J. Plasmeijer, and M. Ronan Sleep. Term graph rewriting. In J. W. de Bakker, A. J. Nijman, and Philip C. Treleaven, editors, *PARLE, Parallel Architectures and Languages Europe, Volume II: Parallel Languages*, volume 259 of *LNCS*, pages 141–158. Springer, 1987. `doi:10.1007/3-540-17945-3`.

7 Adam D. Barwell, Christopher Brown, and Kevin Hammond. Finding parallel functional pearls: Automatic parallel recursion scheme detection in Haskell functions via anti-unification. *Future Generation Comp. Syst.*, 79:669–686, 2018. `doi:10.1016/j.future.2017.07.024`.

8 Alexander Baumgartner. *Anti-Unification Algorithms: Design, Analysis, and Implementation*. PhD thesis, Johannes Kepler University Linz, 2015. Available from `http://www.risc.jku.at/publications/download/risc_5180/phd-thesis.pdf`.

9 Alexander Baumgartner and Temur Kutsia. A library of anti-unification algorithms. In Fermé and Leite [17], pages 543–557. `doi:10.1007/978-3-319-11558-0`.

10 Alexander Baumgartner and Temur Kutsia. Unranked second-order anti-unification. *Inf. Comput.*, 255:262–286, 2017. `doi:10.1016/j.ic.2017.01.005`.

11 Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Higher-order pattern anti-unification in linear time. *J. Autom. Reasoning*, 58(2):293–310, 2017. `doi:10.1007/s10817-016-9383-3`.

12 Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Term-graph anti-unification. RISC Report Series 18-02, Research Institute for Symbolic Computation, Johannes Kepler University Linz, Austria, 2018.

13 Tarek Richard Besold and Enric Plaza. Generalize and blend: Concept blending based on generalization, analogy, and amalgams. In Hannu Toivonen, Simon Colton, Michael

Cook, and Dan Ventura, editors, *Proceedings of the Sixth International Conference on Computational Creativity*, pages 150–157. computationalcreativity.net, 2015. URL: `http://computationalcreativity.net/iccc2015/?page_id=331`.

14 Petr Bulychev and Marius Minea. An evaluation of duplicate code detection using anti-unification. In *Proc. 3rd International Workshop on Software Clones*, 2009.

15 Andrea Corradini and Fabio Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures*, 7(4):299–331, 1999. `doi:10.1023/A:1008647417502`.

16 Rylan Cottrell, Joseph J. C. Chang, Robert J. Walker, and Jörg Denzinger. Determining detailed structural correspondence for generalization tasks. In Ivica Crnkovic and Antonia Bertolino, editors, *6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Int. Symposium on Foundations of Software Engineering*, pages 165–174. ACM, 2007.

17 Eduardo Fermé and João Leite, editors. *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*. Springer, 2014. `doi:10.1007/978-3-319-11558-0`.

18 Adelaine Gelain, Cristiano D. Vasconcellos, Carlos Camarão, and Rodrigo Ribeiro. Type inference for GADTs and anti-unification. In Alberto Pardo and S. Doaitse Swierstra, editors, *Programming Languages - 19th Brazilian Symposium SBLP 2015*, volume 9325 of *LNCS*, pages 16–30. Springer, 2015. `doi:10.1007/978-3-319-24012-1`.

19 Richard Kennaway, Jan Willem Klop, M. Ronan Sleep, and Fer-Jan de Vries. On the adequacy of graph rewriting for simulating term rewriting. *ACM Trans. Program. Lang. Syst.*, 16(3):493–523, 1994. `doi:10.1145/177492.177577`.

20 Jan Willem Klop. Term graph rewriting. In Gilles Dowek, Jan Heering, Karl Meinke, and Bernhard Möller, editors, *Higher-Order Algebra, Logic, and Term Rewriting, Second International Workshop, HOA'95, Selected Papers*, volume 1074 of *LNCS*, pages 1–16. Springer, 1995. `doi:10.1007/3-540-61254-8`.

21 Boris Konev and Temur Kutsia. Anti-unification of concepts in description logic EL. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Fifteenth International Conference, KR'16*, pages 227–236. AAAI Press, 2016. URL: `http://www.aaai.org/Library/KR/kr16contents.php`.

22 Temur Kutsia, Jordi Levy, and Mateu Villaret. Anti-unification for unranked terms and hedges. *J. Autom. Reasoning*, 52(2):155–190, 2014.

23 Giorgio Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9(4):341, 1973.

24 James J McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software: Practice and Experience*, 12(1):23–34, 1982.

25 Santiago Ontañón and Ali Shokoufandeh. Refinement-based similarity measures for directed labeled graphs. In Ashok K. Goel, M. Belén Díaz-Agudo, and Thomas Roth-Berghofer, editors, *Case-Based Reasoning Research and Development - 24th International Conference, ICCBR'16*, volume 9969 of *LNCS*, pages 311–326. Springer, 2016. `doi:10.1007/978-3-319-47096-2`.

26 Gordon D. Plotkin. A note on inductive generalization. *Machine Intell.*, 5(1):153–163, 1970.

27 Detlef Plump. Term graph rewriting. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, chapter 1, pages 3–61. World Scientific, 1999.

28 Simon J. Thompson, Huiqing Li, and Andreas Schumacher. The pragmatics of clone detection and elimination. *Programming Journal*, 1(2):8, 2017. doi:10.22152/programming-journal.org/2017/1/8.