Electronic Communications of the EASST
Volume 71 (2015)

Graph Computation Models
Selected Revised Papers from GCM 2014

Graph Transformation with Symbolic Attributes via Monadic Coalgebra Homomorphisms

Wolfram Kahl

17 pages

Guest Editors: Rachid Echahed, Annegret Habel, Mohamed Mosbah
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Graph Transformation with Symbolic Attributes via Monadic Coalgebra Homomorphisms

Wolfram Kahl*

McMaster University, Hamilton, Ontario, Canada,

Abstract: We show how a coalgebraic approach leads to more natural representations of many kinds of graph structures that in the algebraic approach are frequently dealt with using ad-hoc constructions. For the case of symbolically attributed graphs, we demonstrate how using substituting coalgebra homomorphisms in double-pushout rewriting steps yields a powerful and easily understandable transformation mechanism.

Keywords: Transformation of symbolically attributed graphs, attributed graphs as coalgebras, categoric approach to graph transformation

1 Introduction

An attributed graph is a graph where (some of) the items (nodes and edges) carry "attributes", which are taken from some attribute datatypes.

Formal treatments of datatypes [EM85, BKL+91, BM04] typically characterise datatypes as *algebras*, or "sets with operations"; they are most frequently implemented as software libraries where the sets are only abstract entities, the operations are executable code, and only the elements of the sets are represented as static data.

Graphs, too, can be characterised as algebras, most prominently in the "algebraic approach to graph transformation" [CMR+97, EHK+97, EEPT06]. However, the sets in question are the sets of nodes and edges, and the "operations" are the incidence relations; the whole algebra, understood as a graph, is typically represented as static data.

In attributed graphs, these two conflicting views of algebras come together, and formalisations that consider an attributed graph as a single algebra that includes both graph item sorts and attribute value sorts do not correspond to the way attributed graphs are understood in terms of data organisation. For graph transformation, the theory of the algebraic approach also contributes to the necessity of keeping the graph algebra separate from the attribute value algebra, since pushouts of graph structures, customarily considered as unary algebras [Löw90, CMR+97], can be calculated component-wise, while for typical attribute value algebras, this is not the case. Indeed, most applications have no need to transform the attribute value algebras, since most transformation concepts for attributed graphs expect the transformation results to be attributed over the same attribute datatypes. An exception to this consideration are symbolic attributes, which can easily be drawn from term algebras over different variable sets during different stages of transformation.

* This research is supported by the National Science and Engineering Research Council of Canada, NSERC.

Unary algebras are in fact also co-unary coalgebras, and many kinds of graphs that do not fit the mould of unary algebras can actually naturally be considered as more general coalgebras. This argument was first made in [Kah14]; in the current paper we continue that development and show how to use substituting coalgebra homomorphisms for DPO rewriting of symbolically attributed graphs.

After discussing related work in the next section and providing necessary notation in Sect. 3, we explain the basic technicalities for modelling graph structures using coalgebras in Sect. 4. Using the example of edge-labelled and node-attributed graphs, we move to substituting coalgebra homomorphisms in Sect. 5. The resulting category is an instance of the monadic product coalgebra categories introduced in [Kah14]; we summarise definition and basic results in Sect. 6. The resulting pushouts are used in Sect. 7 to obtain direct derivations of attributed graphs. We contrast our approach with the adhesive approach of [EEPT06] in more detail in Sect. 8. Since the rules and direct derivations of Sect. 7 are not using monomorphisms where most of the current DPO graph transformation literature, and in particular the adhesive approach, prescribe them, we prove a characterisation of monomorphisms in categories of substitutions in Appendix A.

2 Related Work

Löwe *et al.* [LKW93] appear to have been the first to consider attributed graphs in the context of the algebraic approach to graph transformation; they propose to extend the customary unary graph structure signature with an arbitrary attribute signature, and a set of unary attribution operators connecting the two. These attribution operators typically may have as their source special sorts of attribute carriers, which can be deleted and re-created for relabelling.

König and Kozioura [KK08] essentially follow the approach of [LKW93], but choose a rigid organisation of unlabelled nodes, and labelled hyperedges with a single attribute the sort of which is determined by the edge label. Homomorphisms include algebra homomorphisms. In a rule (L,R,α,g), the two rule sides L and R are attributed graphs over the term algebra over a globally fixed set of variables, with L attributed only with variables, and only variables occurring in L may occur in R. The rule morphism is defined by an injective node mapping α; rule morphisms are not defined for edges and attributes, and therefore are a special case of partial morphisms. (The Boolean guard term g controls applicability of the rule.) Matches need to be injective on edges; rewrite steps preserve the data algebra.

In the double-pushout approach, [HKT02, EPT04] use attribution edges connecting graph items with attribute values, and are essentially predecessors of [EEPT06], the approach of which is discussed in more detail in Sect. 8.

All the above consider arbitrary attribute algebras for the application graphs, with term algebras a special case.

For the "symbolic graphs" of [Ore11, OL10b], the data algebra is not considered an explicit part of the graph structure; instead, a "symbolic graph" is an E-graph over a sorted variable set together with a set of formulae (most typically equations) that may refer to constants drawn from the data algebra.

Since the conventional $\mathcal{M}$-adhesive approach does not cover rule applications that change attributes, [Gol12] presents a variant of adhesive categories that softens the adhesive restrictions

to only affect the pushouts that are actually needed during transformation, avoiding spurious non-unifiability problems for attributes. Similarly, Habel and Plump [HP12] restrict the class of morphisms to be used in "vertical" roles in the rewriting steps, to be able to capture the relabelling DPO graph transformations of [HP02, Plu09] which use partially labelled interface graphs.

A different approach to relabelling is that of Rebout [RFS08], which combines de-facto-partial attribution relations with a special mechanism for relabelling via "computations" in the left-hand side of the rule.

Rutten's overview article [Rut00] is useful for general theory of coalgebras. Related with our current work is the part of the coalgebra literature that deals with combining algebras and coalgebras; one approach considers separate algebraic and coalgebraic structures in the same carriers, for example Kurz and Hennicker's "Institutions for Modular Coalgebraic Specifications" [KH02]. A further generalisation are "dialgebras" [Hag87, PZ01], which have a single carrier X, and operations $f_i : F_i X \to G_i X$, where both F_i and G_i are polynomial functors.

3 Notation and Background: Categories and Monads

We assume familiarity with the basics of category theory; for notation, we write "$f : A \to B$" to declare that morphism f goes from object $\mathcal{A}$ to object $\mathcal{B}$, and use ";" as the associative binary *forward composition* operator that maps two morphisms $f : A \to B$ and $g : B \to C$ to $(f ; g) : A \to C$. The identity morphism for object A is written $\mathbb{I}_A$. We assign ";" higher priority than other binary operators, and assign unary operators higher priority than all binary operators.

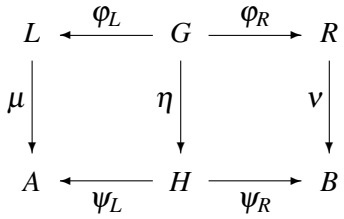
The category of sets and functions is denoted by *Set*.

A *functor* $\mathcal{F}$ from one category to another maps objects to objects and morphisms to morphisms respecting the structure generated by $\to$, $\mathbb{I}$, and composition; we denote functor application by juxtaposition both for objects, $\mathcal{F}\, A$, and for morphisms, $\mathcal{F}\, f$. Although we use forward composition of morphisms, we use backward composition "$\circ$" for functors, with $(\mathcal{G} \circ \mathcal{F})\ A = \mathcal{G}\ (\mathcal{F}\, A)$.

A *monad* on a category $\mathcal{C}$ consists is a functor $\mathcal{M} : \mathcal{C} \to \mathcal{C}$ for which there are two natural transformations ("polymorphic morphisms") $\mathsf{return}_A : A \to \mathcal{M}\ A$ and $\mathsf{join}_A : \mathcal{M}\ (\mathcal{M}\ A) \to \mathcal{M}\ A$ satisfying $\mathsf{return}_{\mathcal{M}\ A} ; \mathsf{join}_A = \mathbb{I}$ and $\mathcal{M}\ \mathsf{return}_A ; \mathsf{join}_A = \mathbb{I}$ and $\mathcal{M}\ \mathsf{join}_A ; \mathsf{join}_A = \mathsf{join}_{\mathcal{M}\ A} ; \mathsf{join}_A$. Important monads are the List monad, and the term monad $\mathcal{T}_\Sigma$ for any (algebraic) signature Σ. For the former, $\mathsf{join}_{\mathsf{List},A} : \mathsf{List}\ (\mathsf{List}\ A) \to \mathsf{List}\ A$ is the function that flattens (or concatenates) lists of lists. For the latter, $\mathcal{T}_\Sigma\ V$ is the set of terms with elements of set V used as variables; the function $\mathsf{join}_{\mathcal{T}_\Sigma, V} : \mathcal{T}_\Sigma\ (\mathcal{T}_\Sigma\ V) \to \mathcal{T}_\Sigma\ V$ maps nested terms (or terms using V-terms as variables) into "flattened" V-terms. Each monad $\mathcal{M}$ on $\mathcal{C}$ induces the so-called *Kleisli category* $\mathbb{K}_\mathcal{M}$ that has the same objects as $\mathcal{C}$, but $\mathcal{C}$-morphisms $A \to \mathcal{M}\ B$ as morphisms from A to B. Kleisli-composition of $f : A \to \mathcal{M}\ B$ with $g : B \to \mathcal{M}\ C$ will be written $f \fatsemi g$; this is defined by $f \fatsemi g = f ; (\mathcal{M}\ g) ; \mathsf{join}_C$.

In the term monad $\mathcal{T}_\Sigma$, Kleisli morphisms are substitutions $\sigma : V_1 \to \mathcal{T}_\Sigma\ V_2$, and Kleisli composition is just composition of substitutions. *Application* of a substitution $\sigma : V_1 \to \mathcal{T}_\Sigma\ V_2$ to a term $t : \mathcal{T}_\Sigma\ V_1$ will be written $\sigma \triangleright t$. (Appendix A contains a few more details about the term monad.)

The double-pushout (DPO) approach to high-level rewriting [CMR+97] uses transformation rules that are spans $L \xleftarrow{\varphi_L} G \xrightarrow{\varphi_R} R$ in an appropriate category between the **l**eft-hand side L, **g**luing object G, and **r**ight-hand side R. A direct transformation step from object A to object B via such a rule is given by a double pushout diagram, with **h**ost object H, where the morphism μ is called the **m**atch.



4 The Coalgebra View of Graph Structures

In the context of the algebraic approach to graph transformation, graph structures have traditionally been presented as unary algebras [Löw90, CMR+97]. However, as such they are the intersection between algebras and coalgebras, and in [Kah14], we showed how more general coalgebras are useful in modelling graph features. Recall: Given a (unary) functor F,

- an F-algebra $A = (C_A, f_A)$ is an object C_A together with a morphism $f_A : F\ C_A \to C_A$
- an F-coalgebra $A = (C_A, f_A)$ is an object C_A together with a morphism $f_A : C_A \to F\ C_A$.

Whereas non-unary algebras allow structured types for the arguments of their operations, non-unary coalgebras allow structured types for their results. Also, while in practical algebras, the shape of the arguments can typically be described by a polynomial functor, more general functors are routinely considered for the shape of the results in coalgebras.

In the signatures for such coalgebras, we therefore allow additional syntax for such functors, like List, with fixed interpretation, just like the product functor $\times$ that is used for the argument shapes of non-unary algebras. In general, we assume a language of functor symbols (with arity), and a *signature* introduces first, after "**sorts:**", a list of *sort symbols*, and then, after "**ops:**", a list of *function symbols* (or *operation symbols*), and for each operation symbol, an argument type expression and a result type expression (separated by "→") each built from the functor symbols and the sort symbols.

- An *algebraic signature* has only single sort symbols as *result* types.
- An *coalgebraic signature* has only single sort symbols as *argument* types.

For example, the following is a coalgebraic signature for directed hypergraphs where each hyperedge has a sequence of source nodes and a sequence of target nodes, and each node is labelled

with an element of the constant set L:

$$\begin{array}{lll} \mathsf{sigDHG} := \langle & \textbf{sorts:} & \mathsf{N}, \mathsf{E} \\ & \textbf{ops:} & \mathsf{src} : \mathsf{E} \to \mathsf{List\ N} \\ & & \mathsf{trg} : \mathsf{E} \to \mathsf{List\ N} \\ & & \mathsf{nlab} : \mathsf{N} \to L \quad \rangle \end{array}$$

The coalgebra functor corresponding to sigDHG is a functor between product categories because of the two sorts:

$$F_{\mathsf{sigDHG}}\ (N\ ,\ E)\ =\ (L\ ,\ ((\mathsf{List}\ N) \times (\mathsf{List}\ N)))$$

Since in algebras, all operations must have a *sort* as result, modelling labelled graphs as algebras always has to employ the trick of declaring the label sets as additional sorts, and then consider the subcategory that has algebras with a fixed choice for these label sets, and morphisms that map them only with the identity. Similarly, list-valued source and target functions are frequently considered for algebraic graph transformation, but with ad-hoc definitions for morphisms and custom proofs of their properties. In contrast, declaring these features via a coalgebra signature such as sigDHG directly captures the mathematical intent.

5 Attributed Graphs as Coalgebras

The expressive power of coalgebraic signatures extends to attributed graphs without any effort. For example, the following is a coalgebraic signature for edge-labelled (with label set $\mathcal{L}$) and node-attributed graphs, with symbolic attributes taken from the term algebra over some term signature Σ and with variables from the variable carrier set for sort V:

$$\begin{array}{lll} \mathsf{sigSNAG}_\Sigma := \langle & \textbf{sorts:} & \mathsf{N}, \mathsf{E}, \mathsf{V} \\ & \textbf{ops:} & \mathsf{src} : \mathsf{E} \to \mathsf{N} \\ & & \mathsf{trg} : \mathsf{E} \to \mathsf{N} \\ & & \mathsf{lab} : \mathsf{E} \to \mathcal{L} \\ & & \mathsf{attr} : \mathsf{N} \to \mathcal{T}_\Sigma\ \mathsf{V} \quad \rangle \end{array}$$

The resulting homomorphism concept only allows renaming of variables:

Fact 5.1 *A* $\mathsf{sigSNAG}_\Sigma$*-coalgebra homomorphism* $F : G_1 \to G_2$ *consists of three mappings* $F_\mathsf{N} : \mathsf{N}_1 \to \mathsf{N}_2$ *and* $F_\mathsf{E} : \mathsf{E}_1 \to \mathsf{E}_2$ *and* $F_\mathsf{V} : \mathsf{V}_1 \to \mathsf{V}_2$ *satisfying the following conditions:*

$$\begin{array}{llll} F_\mathsf{E} ; \mathsf{src}_2 = \mathsf{src}_1 ; F_\mathsf{N} & & F_\mathsf{E} ; \mathsf{lab}_2 = \mathsf{lab}_1 & \\ F_\mathsf{E} ; \mathsf{trg}_2 = \mathsf{trg}_1 ; F_\mathsf{N} & & F_\mathsf{N} ; \mathsf{attr}_2 = \mathsf{attr}_1 ; \mathcal{T}_\Sigma\ F_\mathsf{V} & \square \end{array}$$

DPO rewriting in this category would have to rely on deletion and re-creation of attribute carrying nodes to implement relabelling, similar to [LKW93, KK08]. In addition we also lack the ability to instantiate rules via variable substitution as part of the morphism concept, and might therefore be tempted to add such instantiation outside the DPO rewriting framework, as in [PS04].

The homomorphism concept for $\mathsf{sigSNAG}_\Sigma$-coalgebras can be "fixed" to allow substitution, by also adapting the morphism conditions to take the substituted variables inside the image terms of the attribution function into account:

Definition 5.2 We define the category SNAG_Σ to have $\mathsf{sigSNAG}_\Sigma$-coalgebras as objects, and a morphism $F : G_1 \to G_2$ consists of three mappings typed as shown to the left, satisfying the conditions shown to the right:

$$\begin{array}{lcl} F_\mathsf{N} & : & \mathsf{N}_1 \to \mathsf{N}_2 \\ F_\mathsf{E} & : & \mathsf{E}_1 \to \mathsf{E}_2 \\ F_\mathsf{V} & : & \mathsf{V}_1 \to \mathcal{T}_\Sigma\, \mathsf{V}_2 \end{array} \qquad \begin{array}{lcl} F_\mathsf{E} ; \mathsf{src}_2 & = & \mathsf{src}_1 ; F_\mathsf{N} \\ F_\mathsf{E} ; \mathsf{trg}_2 & = & \mathsf{trg}_1 ; F_\mathsf{N} \end{array} \qquad \begin{array}{lcl} F_\mathsf{E} ; \mathsf{lab}_2 & = & \mathsf{lab}_1 \\ F_\mathsf{N} ; \mathsf{attr}_2 & = & \mathsf{attr}_1 \fatsemi F_\mathsf{V} \end{array}$$

□

Note that F_V is now a morphism in the Kleisli category of the term monad $\mathcal{T}_\Sigma$, and accordingly the homomorphism condition for F_V employs Kleisli composition $\fatsemi$. It is not hard to verify that this category is well-defined — the key to the proof is to recognise that the F_V components are substitutions and compose via Kleisli composition of the term monad.

6 Monadic Product Coalgebras

In [Kah14], we introduced the concept of "monadic product coalgebras" as abstract setting for graph structures with substituting homomorphisms, which distinguishes "graph item sorts" from "variable sorts" by setting the formalisation in the product category $\mathcal{C}_1 \times \mathcal{C}_2$, assuming:

- two categories $\mathcal{C}_1$ and $\mathcal{C}_2$,
- a monad $\mathcal{M}$ on $\mathcal{C}_2$,
- a functor $\mathcal{F}$ from $\mathcal{C}_1 \times \mathcal{C}_2$ to $\mathcal{C}_1$.

In terms of coalgebraic signatures, this implements the restriction that sorts mentioned as monad arguments do not occur as source sorts of operators, and that the monad must not depend on sorts that do occur as source sorts of operators. This restriction is satisfied by all simple kinds of symbolically attributed graphs where the monad is typically a term monad, is applied only to sets of free variables, and these variables do not otherwise participate in the graph structure.

Definition 6.1 ([Kah14]) An *$\mathcal{M}$-$\mathcal{F}$-product-coalgebra* A is a triple $(A_1, A_2, \mathsf{op}_A)$ consisting of

- an object A_1 of $\mathcal{C}_1$, and
- an object A_2 of $\mathcal{C}_2$, and
- a morphism $\mathsf{op}_A : A_1 \to \mathcal{F}\,(A_1,\ \mathcal{M}\, A_2)$

A *$\mathcal{M}$-$\mathcal{F}$-product-coalgebra homomorphism* f from $(A_1, A_2, \mathsf{op}_A)$ to $(B_1, B_2, \mathsf{op}_B)$ is a pair (f_1, f_2) consisting of a $\mathcal{C}_1$-morphism f_1 from A_1 to B_1 and a morphism f_2 from A_2 to B_2 in the Kleisli category of $\mathcal{M}$ such that

$$f_1 ; \mathsf{op}_B = \mathsf{op}_A ; \mathcal{F}\,(f_1,\ \mathcal{M} f_2 ; \mathsf{join}) \quad .$$

Morphism composition is composition of the corresponding product category. □

This morphism composition is well-defined, and induces a category. If we let $\mathcal{M}_0$ be the product monad of the identity monad on $\mathcal{C}_1$ and $\mathcal{M}$, then we see that $\mathcal{M}$-$\mathcal{F}$-product-coalgebra homomorphism are in fact morphisms of the Kleisli category $\mathbb{K}_{\mathcal{M}_0}$, and also use its composition. If we further define $\mathcal{F}_0$ as endofunctor on $\mathcal{C}_1 \times \mathcal{C}_2$ by $\mathcal{F}_0(X_1,X_2) = (\mathcal{F}(X_1,X_2), \mathbb{1})$, then an $\mathcal{M}$-$\mathcal{F}$-product-coalgebra is indeed a $(\mathcal{F}_0 \circ \mathcal{M}_0)$-coalgebra. (This factorisation is further explored in [Kah14], and is too general for pushout creation).

Example 6.2 *The category* SNAG_Σ *of Def. 5.2 is equivalent to the category of* $\mathcal{T}_\Sigma$-$\mathcal{F}_{\mathsf{sigSNAG}}$-*product-coalgebras, where* $\mathcal{C}_1 = \mathrm{Set} \times \mathrm{Set}$ *for nodes and edges,* $\mathcal{C}_2 = \mathrm{Set}$ *for variables (or terms), and*

$$\mathcal{F}_{\mathsf{sigSNAG}}\left((N,E),T\right) = \left(T,(N \times N \times \mathcal{L})\right) \ .$$

The four constituents of the result type of $\mathcal{F}_{\mathsf{sigSNAG}}$ *correspond to the four operators* attr, src, trg, *and* lab *of* sigSNAG, *with* attr *being the first constituent, since it is the only operator starting from sort* N, *while the remaining three all start from sort* E.

Since $\mathcal{M}_0$ is a product monad, pushouts in $\mathbb{K}_{\mathcal{M}_0}$ are calculated component-wise, that is, they consist of a pushout in $\mathcal{C}_1$ and a pushout in the Kleisli category of $\mathcal{M}$, and we have:

Theorem 6.3 ([Kah14]) *The forgetful functor from the category of* $\mathcal{M}$-$\mathcal{F}$-*product-coalgebra homomorphisms to the Kleisli category of* $\mathcal{M}_0$ *creates pushouts.* □

More explicitly, if a span $B \xleftarrow{f} A \xrightarrow{g} C$ of $\mathcal{M}$-$\mathcal{F}$-product-coalgebra homomorphisms is given, and also a cospan $(B_1,B_2) \xrightarrow{h} (D_1,D_2) \xleftarrow{k} (C_1,C_2)$ in $\mathbb{K}_{\mathcal{M}_0}$ that is a pushout for the Kleisli morphisms underlying F and G, then (D_1,D_2) can be extended to a $\mathcal{M}$-$\mathcal{F}$-product-coalgebra $D = (D_1,D_2,\mathsf{op}_D)$ such that $B \xrightarrow{h} D \xleftarrow{k} C$ is a pushout for $B \xleftarrow{f} A \xrightarrow{g} C$ in the $\mathcal{M}$-$\mathcal{F}$-product-coalgebra category.

Together with the equivalence of categories of Example 6.2, pushouts for node-attributed graphs essentially reduce to unification for their variable components (due to the fact that *Set* as underlying category has pushouts):

Corollary 6.4 *A span* $B \xleftarrow{F} A \xrightarrow{G} C$ *in the category* SNAG_Σ *of node-attributed graphs (as* $\mathsf{sigSNAG}_\Sigma$ *structures) has a pushout if* F_V *and* G_V, *as substitutions, have a pushout.* □

7 DPO Transformation with Substituting Homomorphisms

Most DPO approaches to attributed graph transformation insist that the "data algebra" supplying the attribute values remains unchanged by transformation. In contrast, our approach has the data algebra generated by a monad from selected carrier sets — most typically, the data algebra is the term algebra of the variable carrier set. And since variables are just elements of one of the carrier sets, adding and deleting variables is as easy as adding and deleting nodes and edges.

For the sake of readability, we limit the discussion in this section to the symbolically attributed graphs of Sect. 5, but it obviously applies to arbitrary $\mathcal{M}$-$\mathcal{F}$-product-coalgebra categories.

For rewriting of symbolically attributed graphs, we organise the variable set V_G of the gluing graph as a coproduct $V_G = T_G + R_G$ of

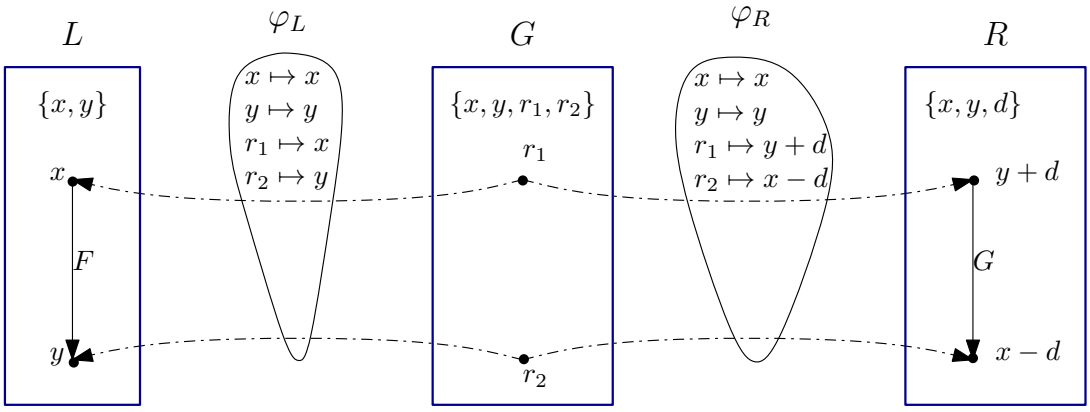
- the set T_G of *transfer variables*, and
- the set R_G of *replacement variables*.

We demand that

- the graph parts (node and edge components) of the rule morphisms are injective,
- the rule morphisms map transfer variables injectively to variables,
- all replacement variables occur in attributes of gluing graph items.

A (rule) morphism satisfying these conditions is called *rigid*.

For human-oriented presentation, and for simplifying the technical arguments below, the transfer-variable parts of the rule morphisms, namely $\varphi_{L,T} : T_G \to \mathcal{T}_\Sigma\, V_L$ and $\varphi_{R,T} : T_G \to \mathcal{T}_\Sigma\, V_R$, will be subset inclusions, with $T_G = V_L \cap V_R$. In the following example drawings, we explicitly list the variable set for each graph, and the variable component (substitution) for each homomorphism.



In the rule drawn above, the transfer variables are x, and y, and the replacement variables are r_1 and r_2; the latter are mapped to different terms on the two rule sides, thus implementing "re-attribution". Furthermore, variable d is "added by the RHS"; if the host graph H had already contained a variable d, then the d of the RHS would have been mapped to some fresh variable in the result graph B.

Note that φ_L is *not* a monomorphism in the category SNAG_Σ of Def. 5.2: Consider a graph Z with empty node and edge sets and with variable set $\{z\}$, and homomorphisms

- $\lambda_1 : Z \to G$ with $\lambda_{1,\mathsf{V}}(z) = x$ and
- $\lambda_2 : Z \to G$ with $\lambda_{2,\mathsf{V}}(z) = r_1$,

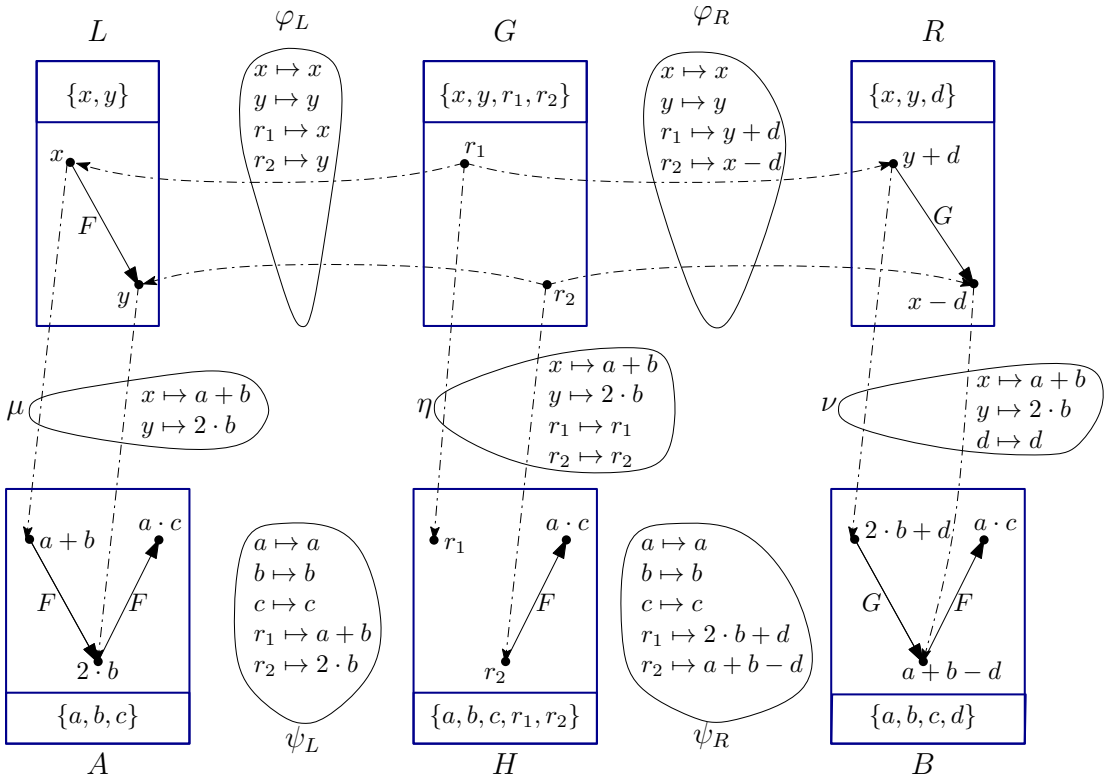
then $\lambda_1 \fatsemi \varphi_L = \lambda_2 \fatsemi \varphi_L$, but $\lambda_1 \neq \lambda_2$. (The homomorphism φ_R "accidentally" happens to be a monomorphism, but it would not be one if, e.g., "$x - d$" had been replaced with "$x - 1$". See Appendix A for more information about monomorphisms in categories of substitutions.)

Although the replacement variables in the example above correspond to undefined labelling in, e.g., [HP12], this is not their only possible use; replacement variables can also occur deeper in the term structure of graph item attributes. This could be used for example to emulate multiple attributes via record-valued single attributes, and then replacing selected attributes could employ gluing nodes with attributes like "$pair(x, r_1)$" or even "$\langle a_1 \mapsto x, a_2 \mapsto r_1, a_3 \mapsto 3 \cdot r_2 \rangle$".

Existence of a pushout complement in the category SNAG_Σ requires, besides the conventional gluing condition for the graph part, the following additional clause for the attribution part:

Definition 7.1 (Variable gluing condition) Each deleted variable (i.e., each variable in $V_L - V_G$) is mapped by the matching μ to a variable in A that does not occur in attributes outside the image of the deleted part of L (which is the "dangling" aspect), and also does not occur in the result of μ_V for any other variable (which is the "identification" aspect). □

Since variable deletion is probably a relatively rarely-useful operation, we show an example redex and DPO transformation step not involving variable deletion on the left-hand side, and therefore trivially satisfying the variable gluing condition (we do not indicate the obvious node and edge mappings for the application span $A \longleftarrow H \longrightarrow B$):



Recall that μ_V is a function of type $V_L \to \mathcal{T}_\Sigma\, V_A$, that is, a substitution that maps variables from V_L to terms over V_A. We will apply it to terms $t : \mathcal{T}_\Sigma\, V_L$ using the notation $\mu_V \triangleright t$ introduced in Sect. 3.

The pushout complement, consisting of the host graph H and the morphisms η and ψ_L, is

obtained via the following steps:

- The graph part (nodes and edges) is constructed as the pushout complement of the graph part of $G \xrightarrow{\varphi_L} L \xrightarrow{\mu} A$.
- We then calculate a least unifier $\gamma : R_G \to \mathcal{T}_\Sigma\, R_G$ that simultaneously unifies (without instantiating any variables of A) all pairs

$$\big(\, \mu_\mathsf{V} \triangleright (\mathsf{attr}_G(n_1)),\quad \mu_\mathsf{V} \triangleright (\mathsf{attr}_G(n_2)) \,\big)$$

  for different preimages $n_1 \neq n_2 \in N_G$ of nodes identified by μ, that is, with $\mu_\mathsf{N}(\varphi_{L,\mathsf{N}}(n_1)) = \mu_\mathsf{N}(\varphi_{L,\mathsf{N}}(n_2))$. Such a least unifier exists since the matching μ proves unifiability.
- The variable set V_H is the disjoint union of the preserved variables of A, that is, $V_P := V_A - \mu_\mathsf{V}(V_L - V_G)$, with the replacement variables of R_G that occur in the range of γ. (Variables that have been unified away must be removed.)
- For the attribution function, we then define (note that γ and μ_V replace disjoint sets of variables):

$$\mathsf{attr}_H(n) = \begin{cases} \gamma\,(\mu_\mathsf{V} \triangleright (\mathsf{attr}_G(m))) & \text{if } n = \eta_\mathsf{N}(m) \text{ with } m \in N_G \\ \mathsf{attr}_A(\psi_{L,\mathsf{N}}(n)) & \text{if } \psi_{L,\mathsf{N}}(n) \notin \mu_\mathsf{N}(N_L) \end{cases}$$

- The substitution η_V is the identity on replacement variables, and coincides with μ_V on transfer variables.
- The substitution $\psi_{L,\mathsf{V}}$ is the identity on preserved variables, and coincides with $\varphi_{L,\mathsf{V}} \fatsemi \mu_\mathsf{V}$ on replacement variables.

Commutativity $\eta_\mathsf{V} \fatsemi \psi_{L,\mathsf{V}} = \varphi_{L,\mathsf{V}} \fatsemi \mu_\mathsf{V}$ is then trivial; the attribute preservation properties

$$\eta_\mathsf{N} \fatsemi \mathsf{attr}_H = \mathsf{attr}_G \fatsemi \eta_\mathsf{V}$$

and

$$\psi_{L,\mathsf{N}} \fatsemi \mathsf{attr}_A = \mathsf{attr}_H \fatsemi \psi_{L,\mathsf{V}}$$

are trivial when γ is trivial (that is, when μ_N does not identify any nodes), and in general require careful analysis for the different variable sets.

The variable gluing condition (Def. 7.1) is essential to show the universality property of the cospan $L \xrightarrow{\mu} A \xleftarrow{\psi_L} H$; altogether we obtain:

Theorem 7.2 (Existence and uniqueness of the pushout-complement) *For $G \xrightarrow{\varphi_L} L \xrightarrow{\mu} A$ in the category* SNAG*, if φ_L is a rigid morphism, then a pushout complement $G \xrightarrow{\eta} H \xrightarrow{\psi_L} A$ exists iff the extended gluing condition is satisfied. If a pushout complement exists, it is unique up to isomorphism.* □

Since the category SNAG does not have all pushouts, the right-hand side of a rule might contribute additional application conditions. Now we define a SNAG-*transformation rule* to be a span $L \xleftarrow{\varphi_L} G \xrightarrow{\varphi_R} R$ of rigid SNAG-homomorphisms. With that restriction, it is easy to see that right-hand side pushouts always exist at least if the matching μ does not identify any nodes. In the case of node identification via μ, the extended gluing condition is only sufficient for construction and well-definedness of the pushout complement; for the construction of the result graph, the following additional condition is necessary:

Definition 7.3 (Attribute identification condition) There is a unifier $\delta : V_R \to \mathcal{T}_\Sigma\, V_R$ that simultaneously unifies all pairs

$$\big(\, \mu_{\mathsf{V}} \triangleright (\mathsf{attr}_R(\varphi_{R,\mathsf{N}}(n_1))),\quad \mu_{\mathsf{V}} \triangleright (\mathsf{attr}_R(\varphi_{R,\mathsf{N}}(n_2)))\,\big)$$

for different preimages $n_1 \neq n_2 \in N_G$ of nodes identified by μ, that is, with $\mu_{\mathsf{N}}(\varphi_{L,\mathsf{N}}(n_1)) = \mu_{\mathsf{N}}(\varphi_{L,\mathsf{N}}(n_2))$. □

Theorem 7.4 (Existence of direct derivation) *For $A \xleftarrow{\mu} L \xleftarrow{\varphi_L} G \xrightarrow{\varphi_R} R$ in the category* SNAG, *if φ_L and φ_R are rigid morphisms and the attribute identification condition is satisfied in addition to the gluing condition for $G \xrightarrow{\varphi_L} L \xrightarrow{\mu} A$, then the usual double-pushout diagram for a direct derivation from A via the rule $L \xleftarrow{\varphi_L} G \xrightarrow{\varphi_R} R$ and the matching μ can be constructed.* □

8 Comparison with Attributed Graph Transformation in the Adhesive Approach

In the adhesive HLR approach to attributed graph transformation, presented in detail in [EEPT06, Chapters 8–12], each attributed graph contains its own Σ-algebra for attribute values. Attributes are associated with graph items (nodes or edges) through special "attribute edges" that have a graph item as source and an attribute value as target. This has the advantage that the source of a matching needs to have only those attributes defined that are relevant for the matching, but also has the disadvantage that "attribute names" require separate mechanisms for distinguishing attributes belonging to different names (achieved via "typing", i.e., move to a slice category, in [EEPT06, Def. 8.7]), for enforcing existence (achieved via "constraints" in [EEPT06, Section 12.1]), and for enforcing uniqueness (apparently requiring tuning a global parameter of the "constraint" mechanism, see [EEPT06, Example 12.2]).

For the implementation AGG, [EEPT06, p. 308] mentions that "AGG allows neither graphs which are only partially attributed, nor several values for one type. This restriction is natural, [...]. In the theory, this restriction can be expressed by adding [...] constraints [...]".

We agree that "this restriction is natural", and we consider coalgebras a far more natural way to incorporate this restriction into the theory of attributed graphs: Any number of attribution operators can be added to a coalgebraic signature, each of these is then necessarily interpreted (implemented) as a (conceptually separate) total function in all coalgebras for that signature.

The typed attributed graph transformation rules of [EEPT06, Chapter 9] are restricted to a "term algebra with variables" for attributes, and the choice of class $\mathcal{M}$ implies that all three graphs of a rule $L \longleftarrow G \longrightarrow R$ share the same term algebra. Therefore, "[the] definition of the

match $[L \to A]$ already requires an assignment for all variables" [EEPT06, p. 183], including those that one might otherwise consider as "introduced in the RHS".

The fact that all horizontal morphisms are restricted to isomorphisms on the value algebra implies that that algebra cannot be modified by transformations. In particular, if the application graph contains a term algebra, it is not possible to add or delete variables.

9 Conclusion and Outlook

The theory of coalgebras, where operations map carrier set elements to arbitrary types constructed via functors from all carrier sets, provides inherent flexibility for modelling of graph structures that is sorely missing from the theory of unary algebras traditionally employed for this purpose in the "algebraic" approach to graph transformation. In particular, coalgebras can model attributed graph structures effortlessly. In contrast, the non-unary value algebras needed for practical applications of attribution form an alien element in the traditional unary algebras modelling graph structures, and therefore require complex auxiliary constructions to properly capture even the simple fact that attribution is a total function from (e.g.) nodes to attribute values, as explained in the previous section.

While the traditional approach handles substitution (typically as a special case of evaluation) via algebra homomorphisms, we use the approach of [Kah14] to handle substitution via Kleisli composition, by factoring the coalgebra functor over an appropriate monad.

In the current paper, we restricted our attention to the term monad, and therefore only considered symbolic attribution with terms in more detail; due to the fact that the set of variables is one of the carrier sets of our coalgebras, adding and deleting variables via DPO transformations is essentially as easy as adding and deleting nodes or edges. Because of this possibility of adding variables via transformation steps, we obtain a symbolic rewriting system that is closer in spirit to that of [OL10a] than to the point of view of [EEPT06], where additional variables in the RHS need to be instantiated as part of the rule application (and indeed already as part of the matching, for technical reasons).

Even though the Kleisli category of the term monad does not have all pushouts (since not all terms are unifiable), we still managed to obtain a rule concept with an application condition that is an only slightly strengthened gluing condition, and that guarantees that a DPO rewriting step can be constructed. Interestingly, the rule sides, although injective on their graph parts, are *not* monomorphisms in our coalgebra category, so none of the current $\mathcal{M}$-adhesive, $\mathcal{W}$-adhesive [Gol12], or $\mathcal{M}$,$\mathcal{N}$-adhesive [HP12] approaches is directly applicable. The general approach of e.g., [Gol12], should however still be applicable to DPO rewriting in categories of monadic product coalgebras — the concrete instance of a $\mathcal{W}$-adhesive category of attributed graphs presented in [Gol12] (implicitly) uses, for enabling attribute change, a partiality monad, which, like the term monad, is a "guarded monad" [GLD05]; we conjecture that guarded monads might be used to unify the two approaches.

For future work, we therefore hope to identify an appropriate variant of the adhesive HLR approaches that does not require monomorphisms for the rule sides, and still supports typical HLR results. The fully abstract generalisation of the results of Sect. 7 to arbitrary $\mathcal{M}$-$\mathcal{F}$-product-

coalgebra categories will then require monads with membership [FHM93] for the gluing condition.

Besides the DPO-based HLR approaches, we also plan to investigate applying to monadic product coalgebras for example also the sesqui-pushout (SqPO) approach of [CHHK06], which is aplied to attributed graph transformation in [DEPR14]. While the approach of Sect. 7 can only delete variables that are matched to variables, sesqui-pushout rewriting should give us more flexibility there. It may even be advantageous to explore applying the fibred apprach to rewriting [Kah97], as this provies a principled approach to distinguish the different ways substitution and/or partiality are employed in the horizontal versus the vertical arrows of "double-square rewriting".

We also plan to investigate moving from term algebras to more general algebras as target types for attributions.

Bibliography

[BKL+91] M. Bidoit, H.-J. Kreowski, P. Lescanne, F. Orejas, D. Sanella (eds.). *Algebraic System Specification and Devalopment — A Survey and Annotated Bibliography*. LNCS 501. Springer, 1991.

[BM97] R. S. Bird, O. de Moor. *Algebra of Programming*. International Series in Computer Science 100. Prentice Hall, 1997.

[BM04] M. Bidoit, P. D. Mosses. CASL *User Manual*. LNCS (IFIP Series) 2900. Springer, 2004. With chapters by T. Mossakowski, D. Sannella, and A. Tarlecki.
http://link.springer.de/link/service/series/0558/tocs/t2900.htm

[CEKR02] A. Corradini, H. Ehrig, H. Kreowski, G. Rozenberg (eds.). *Proc. First International Conference on Graph Transformation, ICGT 2002*. LNCS 2505. 2002.
doi:10.1007/3-540-45832-8

[CHHK06] A. Corradini, T. Heindel, F. Hermann, B. Knig. Sesqui-Pushout Rewriting. In Corradini et al. (eds.), *Graph Transformations*. LNCS 4178, pp. 30–45. Springer Berlin Heidelberg, 2006.
doi:10.1007/11841883_4
http://dx.doi.org/10.1007/11841883_4

[CMR+97] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, M. Löwe. Algebraic Approaches to Graph Transformation, Part I: Basic Concepts and Double Pushout Approach. Chapter 3, pp. 163–245 in [Roz97].

[DEPR14] D. Duval, R. Echahed, F. Prost, L. Ribeiro. Transformation of Attributed Structures with Cloning. In Gnesi and Rensink (eds.), *Fundamental Approaches to Software Engineering*. LNCS 8411, pp. 310–324. Springer Berlin Heidelberg, 2014.
doi:10.1007/978-3-642-54804-8_22

[EEKR12] H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg (eds.). *Proc. Sixth International Conference on Graph Transformation, ICGT 2012*. LNCS 7562. Springer, 2012.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.

[EHK+97] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, A. Corradini. Algebraic Approaches to Graph Transformation, Part II: Single Pushout Approach and Comparison with Double Pushout Approach. Chapter 4, pp. 247–312 in [Roz97].

[EM85] H. Ehrig, B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer, 1985.

[EPT04] H. Ehrig, U. Prange, G. Taentzer. Fundamental Theory for Typed Attributed Graph Transformation. Pp. 161–177 in [PBE04].
doi:10.1007/b100934

[FHM93] P. Freyd, P. Hoogendijk, O. de Moor. Membership of Datatypes. Dec. 1993. Unpublished manuscript, see also [BM97, Sect. 6.5].

[GLD05] N. Ghani, C. Lüth, F. De Marchi. Monads of Coalgebras: Rational Terms and Term Graphs. *Mathematical Structures in Computer Science* 15:433–451, 2005.
doi:10.1017/S0960129505004743

[Gol12] U. Golas. A General Attribution Concept for Models in $\mathcal{M}$-Adhesive Transformation Systems. Pp. 187–202 in [EEKR12].
doi:10.1007/978-3-642-33654-6_13

[Hag87] T. Hagino. *A Categorical Programming Language*. PhD thesis, Edinburgh University, 1987.

[HKT02] R. Heckel, J. M. Küster, G. Taentzer. Confluence of Typed Attributed Graph Transformation Systems. Pp. 161–176 in [CEKR02].
doi:10.1007/3-540-45832-8_14

[HP02] A. Habel, D. Plump. Relabelling in Graph Transformation. Pp. 135–147 in [CEKR02].
doi:10.1007/3-540-45832-8_12

[HP12] A. Habel, D. Plump. $\mathcal{M}$,$\mathcal{N}$-Adhesive Transformation Systems. Pp. 218–233 in [EEKR12].
doi:10.1007/978-3-642-33654-6_15

[Kah97] W. Kahl. A Fibred Approach to Rewriting — How the Duality between Adding and Deleting Cooperates with the Difference between Matching and Rewriting. Technical report 9702, Fakultät für Informatik, Universität der Bundeswehr München, May 1997. http://www.cas.mcmaster.ca/~kahl/Publications/TR/Kahl-1997b.html.
http://www.cas.mcmaster.ca/~kahl/Publications/TR/Kahl-1997b.html

[Kah14] W. Kahl. Categories of Coalgebras with Monadic Homomorphisms. In Bonsangue (ed.), *Coalgebraic Methods in Computer Science, CMCS 2014*. LNCS 8446, pp. 151–167. Springer, 2014. Agda theories available via http://RelMiCS.McMaster.ca/RATH-Agda/.
doi:10.1007/978-3-662-44124-4_9

[KH02] A. Kurz, R. Hennicker. On Institutions for Modular Coalgebraic Specifications. *Theoretical Computer Science* 280(1–2):69–103, 2002.
doi:10.1016/S0304-3975(01)00021-4

[KK08] B. König, V. Kozioura. Towards the Verification of Attributed Graph Transformation Systems. In Ehrig et al. (eds.), *4th Intl. Conf. on Graph Transformations, ICGT 2008*. LNCS 5214, pp. 305–320. 2008.
doi:10.1007/978-3-540-87405-8_21

[LKW93] M. Löwe, M. Korff, A. Wagner. An Algebraic Framework for the Transformation of Attributed Graphs. In Sleep et al. (eds.), *Term Graph Rewriting: Theory and Practice*. Pp. 185–199. Wiley, 1993.

[Löw90] M. Löwe. Algebraic Approach to Graph Transformation Based on Single Pushout Derivations. Technical report 90/05, TU Berlin, 1990.

[OL10a] F. Orejas, L. Lambers. Delaying Constraint Solving in Symbolic Graph Transformation. In Ehrig et al. (eds.), *Fifth Intl. Conf. on Graph Transformation, ICGT 2010*. LNCS 6372, pp. 43–58. Sept. 2010.
doi:10.1007/978-3-642-15928-2_4

[OL10b] F. Orejas, L. Lambers. Symbolic Attributed Graphs for Attributed Graph Transformation. *ECEASST* 30(Graph and Model Transformation 2010):2.1–2.25, 2010.

[Ore11] F. Orejas. Symbolic Graphs for Attributed Graph Constraints. *J. Symbolic Comput.* 46(3):294–315, Mar. 2011.
doi:10.1016/j.jsc.2010.09.009

[PBE04] F. Parisi-Presicce, P. Bottoni, G. Engels (eds.). *Proc. Second International Conference on Graph Transformation, ICGT 2004*. LNCS 3256. 2004.
doi:10.1007/b100934

[Plu09] D. Plump. The Graph Programming Language GP. In *Algebraic Informatics, CAI 2009*. LNCS 5725, pp. 99–122. 2009.
doi:0.1007/978-3-642-03564-7_6

[PS04] D. Plump, S. Steinert. Towards Graph Programs for Graph Algorithms. Pp. 128–143 in [PBE04].
doi:10.1007/978-3-540-30203-2_11

[PZ01] E. Poll, J. Zwanenburg. From Algebras and Coalgebras to Dialgebras. *ENTCS* 44(1):289–307, 2001.
doi:10.1016/S1571-0661(04)80915-0

[RFS08] M. Rebout, L. Féraud, S. Soloviev. A Unified Categorical Approach for Attributed Graph Rewriting. In Hirsch et al. (eds.), *Computer Science — Theory and Applications, CSR 2008*. LNCS 5010, pp. 398–409. Springer, 2008. doi:10.1007/978-3-540-79709-8_39

[Roz97] G. Rozenberg (ed.). *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*. World Scientific, Singapore, 1997.

[Rut00] J. J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science* 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6

A Monomorphisms in Substitution Categories

Let $\mathcal{T}_\Sigma$ denote the term functor for signature Σ, that is, $\mathcal{T}_\Sigma\, X$ is the set of Σ-terms with elements of set X as variables. Let $\mathsf{FV}(t)$ denote the set of (free) variables occurring in term t.

$\mathcal{T}_\Sigma$ is an endofunctor on the category *Set*, and naturally extends to a monad (see Sect. 3), the *term monad*. Its "join" natural transformation, $\mathsf{join}_{\mathcal{T}_\Sigma}$, produces for each set A (of variables) the function $\mathsf{join}_{\mathcal{T}_\Sigma, A} : \mathcal{T}_\Sigma\,(\mathcal{T}_\Sigma\, A) \to \mathcal{T}_\Sigma\, A$ which "flattens" nested terms over variables in A (that is, terms over $\mathcal{T}_\Sigma\, A$ as their set of variables).

Substitutions, that is, functions $X \to \mathcal{T}_\Sigma\, Y$, are morphisms in the Kleisli category of the term monad $\mathcal{T}_\Sigma$, and therefore compose via Kleisli composition, which is defined for arbitrary substitutions $F : X \to \mathcal{T}_\Sigma\, Y$ and $G : Y \to \mathcal{T}_\Sigma\, Z$ as follows:

$$F \fatsemi G = F ; \mathcal{T}_\Sigma\, G ; \mathsf{join}_{\mathcal{T}_\Sigma, Z}$$

Conventionally, this would be described via "application" of substitutions to terms — since we write $F \triangleright t$ for the application of substitution $F : X \to \mathcal{T}_\Sigma\, Y$ to term $t : \mathcal{T}_\Sigma\, X$, the composition $F \fatsemi G$ can defined by

$$(F \fatsemi G)(v) = F \triangleright (G(v)) \quad , \qquad \text{for all } v : X.$$

When starting from the monadic setting, application of substitutions can be defined as follows:

$$F \triangleright t = ((\mathcal{T}_\Sigma\, F) ; \mathsf{join}_Y)(t)$$

Monomorphisms

In any monad, if the "return" natural transformation produces monomorphisms (which it does for $\mathcal{T}_\Sigma$), then monomorphisms in the Kleisli category of this monad are also monomorphisms in the underlying category. Monomorphisms F of the underlying category that are preserved by the monad functor give rise to monomorphisms $F ; \mathsf{return}$ in the Kleisli category.

The term functor preserves all monomorphisms: An injective variable mapping $F : V_1 \to V_2$ gives rise to an injective term mapping $\mathcal{T}_\Sigma\, F : \mathcal{T}_\Sigma\, V_1 \to \mathcal{T}_\Sigma\, V_2$ that only renames variables. The resulting substitution $F ; \mathsf{return} : V_1 \to \mathcal{T}_\Sigma\, V_2$ is an injective variable renaming, which is therefore a mono in the category of substitutions, too — this also can easily be seen directly.

The category of substitutions has pushouts along such variable renamings; these pushouts implement name-clash-avoiding extension of the domain of substitutions.

For σ, being a monomorphism in the category of substitutions exactly means that *substitution application* of σ does not unify any two different terms. That is, σ is a monomorphism in the category of substitutions iff for any two terms t_1 and t_2 we have

$$\sigma \triangleright t_1 = \sigma \triangleright t_2 \qquad \text{implies} \qquad t_1 = t_2 \ .$$

From this, it is easy to see that monomorphisms in the category of substitutions cannot map any variables to ground terms.

However, this condition is not easy to check directly.

Fortunately a much simpler condition is (necessary and) sufficient: We can show that monomorphisms in the Kleisli category of the term monad are those substitutions that do not identify variables with different terms:

Theorem A.1 *A substitution $\sigma : V_1 \to \mathcal{T}_\Sigma\, V_2$ is a monomorphism in the category of substitutions iff for every variable $v : V_1$ and every term $t : \mathcal{T}_\Sigma\, V_1$, we have:*

$$\sigma\, v = \sigma \triangleright t \qquad \textit{implies} \qquad v = t \ .$$

Proof. "⇒" follows directly by applying the monomorphism property to the two terms v and t. "⇐": Assume that σ satisfies the given condition. To show that σ is a monomorphism in the category of substitutions it suffices to show that for any two terms $t, u : \mathcal{T}_\Sigma\, V_1$ with $\sigma \triangleright t = \sigma \triangleright u$, we have $t = u$. Since this is actually equivalent to restricting V_0 to a one-element set, it suffices to show that for all terms $t_1, t_2 : \mathcal{T}_\Sigma\, V_1$ with $\sigma \triangleright t_1 = \sigma \triangleright t_2$ we have $t_1 = t_2$.

- If $t = v$ is a variable, then $\sigma\, v = \sigma \triangleright t = \sigma \triangleright u$, from which the given property yields $v = u$. (The case where u is a variable is analogous.)
- If $t = f(t_1, \ldots t_n)$ and $u = g(u_1, \ldots u_n)$, then $\sigma \triangleright t = \sigma \triangleright u$ implies $f = g$ and $\sigma \triangleright t_i = \sigma \triangleright u_i$, from which the induction hypothesis yields $t_i = u_i$ for all i, implying $t = u$. □

For finite substitutions, this condition directly translates into a decision procedure that for each variable $v : V_1$ checks whether for any *different* variable $u : V_1$, its image $\sigma\, u$ occurs as a subterm in $\sigma\, v$.