# Categories of Coalgebras with Monadic Homomorphisms

Wolfram Kahl

# Categories of Coalgebras with Monadic Homomorphisms

Wolfram Kahl

McMaster University, Hamilton, Ontario, Canada, kahl@cas.mcmaster.ca

**Abstract.** Abstract graph transformation approaches traditionally consider graph structures as algebras over signatures where all function symbols are unary.
Attributed graphs, with attributes taken from (term) algebras over arbitrary signatures do not fit directly into this kind of transformation approach, since algebras containing function symbols taking two or more arguments do not allow component-wise construction of pushouts.
We show how shifting from the algebraic view to a coalgebraic view of graph structures opens up additional flexibility, and enables treating term algebras over arbitrary signatures in essentially the same way as unstructured label sets. We integrate substitution into our coalgebra homomorphisms by identifying a factoring over the term monad, and obtain a flexible framework for graphs with symbolic attributes. This allows us to prove that pushouts can be constructed for homomorphisms with unifiable substitution components.
We formalised the presented development in Agda, which crucially aided the exploration of the complex interaction of the different functors, and enables us to report all theorems as mechanically verified.

## 1 Introduction

In computer science, algebras are used in two different rôles:

- "Algebras providing datatype" are the concern in particular of the field of algebraic specifications [EM85, BKL$^+$91, BM04]: The carrier sets of an algebra are datatypes, and the operations are available as some kind of executable function. Frequently, the carrier sets are so large that one would not consider to keep all their elements simultaneously available in some data structure.
- "Algebras as data" are most obviously the topic of the algebraic approach to graph transformation [CMR$^+$97, EHK$^+$97, EEPT06] which derives its name from the fact that it considers graphs as algebras.

In *attributed graphs*, the two views come together: An attributed graph is first of all a graph, that is an algebra that is considered in its whole as a piece of data, but (some of) its items may be assigned attributes which are elements of some datatypes provided by an attribute algebra, which is normally not considered in its whole as a piece of data. Although an attributed graph can be

considered as a single algebra, implementation considerations alone already dictate a separation into a graph structure part and an attribution part. As far as attributed graphs are to be transformed via the algebraic approach to graph transformation, theoretical reasons contribute to this separation: for graph structures, considered as unary algebras, the pushouts of their homomorphisms can be calculated component-wise and independent of the presence of operations between the different sorts, while in the presence of non-unary operations, this is no longer the case. With more-than-unary operations, even a pushout of finite algebras can become infinite, so that calculations of these pushouts is in general not feasible. There is also typically little motivation to consider non-trivial pushouts of attribute 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.

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 this paper we show how more general coalgebras are useful in modelling graph features, in particular symbolic attribution. Therefore, we define our graph structures not via algebraic signatures, but via coalgebraic signatures, and integrate label types and term type constructors for attributes into the coalgebraic result types.

For example, the following is a 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 $L$:

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

While constant sets like $L$ are perfectly standard as results in coalgebras, 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 makes the generic theory of coalgebras available, which immediately produces the standard homomorphism definition for directed hypergraphs considered as sigDHG structures, without any necessity for ad-hoc treatment of the label type or the list structure.

Even more interesting is the use of coalgebras for symbolically attributed graphs, where morphisms are required to also contain substitutions for attribute variables; the main contribution of this paper is to formulate the beginnings of a coalgebraic approach to corresponding categories of symbolically attributed graphs.

After discussing related work in the next section, we provide some more detailed motivation for moving to coalgebras. We quickly fix our categorical notation in Sect. 3 and explain basics of (co)algebras in Sect. 4. We show more complex graphs structures in Sect. 5, and discuss the limitations of using standard coalgebra homomorphisms. In section Sect. 6 we show how a factoring of the coalgebra functor over a monad allows us to replace the morphisms underlying the coalgebra homomorphisms with Kleisli arrows, enabling typical applications of symbolic attributes where instantiation of variables via substitution is required and the variable set may be modified by transformations. We show that this general factoring accommodates a natural formalisation of term graphs as monadic coalgebras. Refining this factoring in Sect. 7 for a general class of structures encompassing in particular common kinds of symbolically attributed graphs, we show that pushouts in that setting can be constructed from unifications for the substitution components of the homomorphisms.

The whole theoretical development has been formalised in the dependently typed programming language and proof checker Agda2 [Nor07] on top of the basic category formalisations provided by [Kah11, Kah14]. The Agda source code for this development is available on-line[1]. For not disrupting the flow of the presentation, we just add a check mark " ✓ " to statements for which a formalised version has been mechanically checked by Agda.

## 2 Related Work

Löwe *et al.* [LKW93] started to consider attributed graphs in the context of the algebraic approach to graph transformation; they propose working with a tripartitioned signature, with a unary graph structure part, an arbitrary attribute signature, and a set of unary attribution operators connecting the two. Rewriting uses the single pushout approach. Without discussing the issue in depth, they propose to add sorts of attribute carriers that are deleted and re-created for relabelling. König and Kozioura [KK08] follow the approach of [LKW93], but impose a rigid organisation of unlabelled nodes, and labelled hyperedges with label-conforming attribution.

In the double-pushout approach, Heckel *et al.* [HKT02] treat data algebra carriers as graph nodes, with graph edges to them allowed, but data algebra function symbols are not part of the graph. Their attribution edges from graph nodes to data nodes are equivalent to the attribute carriers of [LKW93]. The data part is kept constant during transformation. Rule graphs are attributed over a term algebra with a fixed set of variables. The E-graphs of [EPT04] allows

[1] URL: http://relmics.mcmaster.ca/RATH-Agda/

also attribute edges starting from edges. The algebra integration of [HKT02] is strengthened from a commuting square to a pullback, which is used for showing the equivalence of categories of typed attributed graphs over type graph *ATG* with categories of algebras over a derived signature *AGSIG(ATG)*, where each type graph item is turned into a sort. For the symbolic graphs of [OL10], the $\Sigma$-algebra is not integrated into the graph structure, but only connected to it via constraints: A symbolic graph is an E-graph over a sorted variable set together with a set of formulae that may refer to constants drawn from the $\Sigma$-algebra.

While all the approaches presented so far worked with total algebras throughout, the relabelling DPO graph transformations of Habel and Plump [HP02, Plu09] use partially labelled interface graphs. Rule side images of unlabelled interface nodes are unlabelled as well, and natural pushouts (that are also pullbacks) with injective matching are used for rewriting. In [PS04], rule schemas are introduced to get around the fixed label sets of [HP02]; these rule schemas are rules that are labelled over a term algebra. A different approach to relabelling is that of Rebout [RFS08], which employs a special mechanism for relabelling via "computations" in the left-hand side of the rule.

For general theory of coalgebra, we refer to Rutten's overview article [Rut00]. The part of the coalgebra literature that deals with combining algebras and coalgebras is probably closest to our current endeavour; 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.

Pardo studies the combination of corecursion with monads [Par98], using as an essential tool natural transformations for distribution of the monad over the functor; his "monadic coalgebras" are defined by an operation of type $A \to \mathcal{M}\ (\mathcal{F}\ A)$, which is the opposite functor composition to the one we use in Sect. 6. Capretta's survey [Cap11] covers coalgebras in functional programming and type theory; like Pardo, also Capretta concentrates on coinduction and infinite structures.

## 3 Category Notation

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 $(f \fatsemi g) : A \to C$. 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.

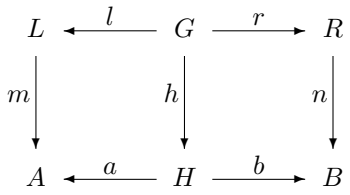
The category of sets and functions is denoted by *Set*.

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 com-

position; 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 $(\mathcal{G} \circ \mathcal{F})\ A = \mathcal{G}\ (\mathcal{F}\ A)$, and may even omit parentheses and just write $\mathcal{G}\mathcal{F}A$.

A *bifunctor* is a functor where the source is a product category. An important example is the coproduct bifunctor $+ : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ for a category $\mathcal{C}$ with a choice of coproducts. Functors with more than two arguments are handled similarly.

The double-pushout (DPO) approach to high-level rewriting [CMR+97]. uses transformation rules that are spans $L \xleftarrow{l} G \xrightarrow{r} R$ in an appropriate category between the left-hand side $L$, gluing object $G$, and right-hand side $R$. A direct transformation step from object $A$ to object $B$ via such a rule is given by a double pushout diagram, where $m$ is called the match:




## 4 Algebras and Coalgebras

The category-theoretic definitions of algebras and coalgebras are simple: Given a (unary) functor $\mathcal{F}$,

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

The algebraic approach to graph transformation was named for its understanding of graphs as algebras — unlabelled graphs are conventionally presented as algebras over the the following signature:

$$\begin{aligned} \mathsf{sigGraph} := \langle\ & \textbf{sorts: } \mathsf{N}, \mathsf{E} \\ & \textbf{ops: } \mathsf{src} : \mathsf{E} \to \mathsf{N} \\ & \qquad\ \mathsf{trg} : \mathsf{E} \to \mathsf{N} \qquad \rangle \end{aligned}$$

(From now on, we assume the product bifunctor $\times$, the coproduct bifunctor $+$, the terminal object $\mathbb{1}$, and the initial object $\mathbb{0}$ to be given.) The functor giving rise to graphs as algebras is a functor on the product category $\mathit{Set} \times \mathit{Set}$ since there is more than one sort:

$$\mathcal{F}_{\mathsf{sigGraph-alg}}\ (N\ ,\ E)\ =\ ((E + E)\ ,\ \mathbb{0})\ ,$$

since there is an isomorphism mapping functions $E+E \to N$ to pairs of functions $(E \to N) \times (E \to N)$, and there is only one "empty" function in $\mathbb{0} \to E$.

This functor can be constructed systematically from the signature above, and the signatures for which this systematic procedure works are called "algebraic":

– An *algebraic signature* has only single sort symbols as *result* types.

Dually, a coalgebra functor can be constructed systematically for the following:

– An *coalgebraic signature* has only single sort symbols as *argument* types.

Obviously, $\mathsf{sigGraph}$ is also a coalgebraic signature, and the functor giving rise to graphs as coalgebras is the following:

$$\mathcal{F}_{\mathsf{sigGraph}}\ (N\ ,\ E)\ =\ (\mathbb{1}\ ,\ (N \times N))$$

For algebras, one frequently considers only *polynomial functors*, that is, functors constructed from $+$, $\times$, and $\mathbb{1}$. For coalgebras, more varied functors are the norm, and many more complicated kinds of graphs can easily be characterised via coalgebraic signatures, for example:

– Node-labelled graphs are often presented with signature $\mathsf{sigNLG}_1$ for some node label set $L$ — note that $\mathsf{sigNLG}_1$ is not an algebraic signature, since $L$ is not a sort symbol:

$$\mathsf{sigNLG}_1 := \langle\ \textbf{sorts:}\ \mathsf{N}, \mathsf{E} \quad \textbf{ops:}\ \mathsf{src} : \mathsf{E} \to \mathsf{N},\ \mathsf{trg} : \mathsf{E} \to \mathsf{N},\ \mathsf{nlab} : \mathsf{N} \to L\ \rangle$$

$$\mathsf{sigNLG}_2 := \langle\ \textbf{sorts:}\ \mathsf{N}, \mathsf{E}, \mathsf{L} \quad \textbf{ops:}\ \mathsf{src} : \mathsf{E} \to \mathsf{N},\ \mathsf{trg} : \mathsf{E} \to \mathsf{N},\ \mathsf{nlab} : \mathsf{N} \to \mathsf{L}\ \rangle$$

Although this is easily fixed, see $\mathsf{sigNLG}_2$ which introduces an additional sort $\mathsf{L}$, this comes at the cost of considering the label set a *part of the graph*, while usually one may want to consider it as fixed. The category of $\mathsf{sigNLG}_2$-structures admits morphisms that change labels, and encompasses as subcategories images of the categories of $\mathsf{sigNLG}_1$-structures for different choices of the interpretation of $L$.
However, both $\mathsf{sigNLG}_1$ and $\mathsf{sigNLG}_2$ are coalgebraic signatures, which shows that the coalgebraic view has advantages even when dealing with very simple graph structures. For a fixed node set $L$, coalgebras over the functor for $\mathsf{sigNLG}_1$ form exactly the category of graphs with node labels drawn from $L$, without any complications:

$$\mathcal{F}_{\mathsf{sigNLG}_1}\ (N, E)\ =\ (L\ ,\ N \times N)$$

– $\mathsf{sigDHG}$, already mentioned in the introduction, is a 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 $L$:

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

Writing $\mathsf{List}$ for the list functor[2], the functor corresponding to $\mathsf{sigDHG}$ is again a functor between product categories because of the two sorts:

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

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 "$\to$") each built from the functor symbols and the sort symbols.

In $\mathsf{sigDHG}$, we used the unary functor symbol $\mathsf{List}$ and the zero-ary functor symbol $L$ — we will not make any notational distinction between functor symbols and their interpretation as functors.

## 5 Limitations of Standard Coalgebra Homomorphisms

For a different situation consider edge-attributed graphs, with symbolic attributes taken from the term algebra $\mathcal{T}_\Sigma\ \mathsf{V}$ over some term signature $\Sigma$ and with variables from the variable carrier set for sort $\mathsf{V}$:

$$\begin{aligned}\mathsf{sigAG}_\Sigma := \langle\ &\textbf{sorts: } \mathsf{N}, \mathsf{E}, \mathsf{V}\\ &\textbf{ops: } \mathsf{src} : \mathsf{E} \to \mathsf{N}\\ &\qquad\ \ \mathsf{trg} : \mathsf{E} \to \mathsf{N}\\ &\qquad\ \ \mathsf{attr} : \mathsf{E} \to \mathcal{T}_\Sigma\ \mathsf{V}\quad \rangle\end{aligned}$$

The resulting homomorphism concept only allows renaming of variables:

**Fact 5.1** A $\mathsf{sigAG}_\Sigma$-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:

$$\begin{aligned}F_\mathsf{E} \fatsemi \mathsf{src}_2 &= \mathsf{src}_1 \fatsemi F_\mathsf{N}\\ F_\mathsf{E} \fatsemi \mathsf{trg}_2 &= \mathsf{trg}_1 \fatsemi F_\mathsf{N}\\ F_\mathsf{E} \fatsemi \mathsf{attr}_2 &= \mathsf{attr}_1 \fatsemi \mathcal{T}_\Sigma\ F_\mathsf{V}\end{aligned}$$

□

DPO rewriting in this category therefore has to rely on deletion and re-creation of attribute carrying edges to implement relabelling, like the approaches of [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].

[2] Note that $\mathsf{List}\ A$ can be defined as the initial algebra of the functor $L_A\ Y = \mathbb{1} + A \times Y$.

Another example where the coalgebra category is unsatisfactory are term graphs, where each node is either a variable (of sort $\mathsf{V}$), or an inner node (of sort $\mathsf{N}$) that has a label (from set $L$) and a list of successors, which can be either variables or other nodes:

$$\begin{aligned}\mathsf{sigTG} := \langle\ & \textbf{sorts: } \mathsf{V}, \mathsf{N} \\ & \textbf{ops: } \mathsf{lab} : \mathsf{N} \to L \\ & \qquad \mathsf{suc} : \mathsf{N} \to \mathsf{List}\ (\mathsf{N} + \mathsf{V}) \quad \rangle\end{aligned}$$

The resulting standard homomorphism concept also has $F_\mathsf{V} : \mathsf{V}_1 \to \mathsf{V}_2$ and therefore does not allow mapping of variables to inner nodes:

**Fact 5.2** A $\mathsf{sigTG}$-coalgebra homomorphism $F : G_1 \to G_2$ consists of two mappings $F_\mathsf{N} : \mathsf{N}_1 \to \mathsf{N}_2$ and $F_\mathsf{V} : \mathsf{V}_1 \to \mathsf{V}_2$ satisfying the following conditions:

$$\begin{aligned} F_\mathsf{N} \mathbin{;} \mathsf{lab}_2 &= \mathsf{lab}_1 \\ F_\mathsf{N} \mathbin{;} \mathsf{suc}_2 &= \mathsf{suc}_1 \mathbin{;} \mathsf{List}(F_\mathsf{N} + F_\mathsf{V}) \end{aligned}$$ □

In the resulting category, pushout complements exist only in very special cases, and the resulting DPO rewriting concept does not correspond to any useful term graph rewriting concept.

## 6 Monadic Coalgebra Morphisms

We now introduce a more powerful morphism concept to remedy these shortcomings. We first show how the homomorphism concepts for $\mathsf{sigAG}_\Sigma$-coalgebras and for $\mathsf{sigTG}$-coalgebras can be "fixed" to allow substitution, and then extract the general pattern behind this class of "fixes".

### 6.1 Substituting Attributed Graph Homomorphisms

If we want to allow substitutions in morphisms between $\mathsf{sigAG}_\Sigma$-coalgebras, we also have to adapt the morphism conditions to take the substituted variables inside the image terms of the attribution function into account:

**Definition 6.1** We define the category $\mathsf{AG}_\Sigma$ to have $\mathsf{sigAG}_\Sigma$-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:

$$\begin{aligned} F_\mathsf{N} &: \mathsf{N}_1 \to \mathsf{N}_2 \\ F_\mathsf{E} &: \mathsf{E}_1 \to \mathsf{E}_2 \\ F_\mathsf{V} &: \mathsf{V}_1 \to \mathcal{T}_\Sigma\ \mathsf{V}_2 \end{aligned} \qquad \begin{aligned} F_\mathsf{E} \mathbin{;} \mathsf{src}_2 &= \mathsf{src}_1 \mathbin{;} F_\mathsf{N} \\ F_\mathsf{E} \mathbin{;} \mathsf{trg}_2 &= \mathsf{trg}_1 \mathbin{;} F_\mathsf{N} \\ F_\mathsf{E} \mathbin{;} \mathsf{attr}_2 &= \mathsf{attr}_1 \mathbin{;} \mathcal{T}_\Sigma\ F_\mathsf{V} \mathbin{;} \mu_{\mathcal{T}_\Sigma} \end{aligned}$$

where $\mu_{\mathcal{T}_\Sigma} : \forall X\ .\ \mathcal{T}_\Sigma(\mathcal{T}_\Sigma\ X) \to \mathcal{T}_\Sigma\ X$ is the canonical "term flattening" function that turns two-level nested terms into one-level terms. □

It is not hard to verify that this category is well-defined✓ — the key to the proof is to recognise that the $F_{\mathsf{V}}$ components are substitutions and compose via Kleisli composition of the term monad.

The category $\mathsf{AG}_\Sigma$ of course does not have all pushouts, since pushout construction for the $F_{\mathsf{V}}$ components involves term unification, which is not always defined.

### 6.2 "Substituting" Term Graph Homomorphisms

For term graphs, we just want to allow variables to be mapped also to inner nodes, and therefore adapt the type of $F_{\mathsf{V}}$ accordingly. The resulting adaptation in the commutativity condition for $\mathsf{suc}$ affects only the argument of the $\mathsf{List}$ functor:

**Definition 6.2** We define the category $\mathsf{TG}$ to have $\mathsf{sigTG}$-coalgebras as objects, and a morphism $F : G_1 \to G_2$ consists of two mappings

$$F_{\mathsf{N}} : \mathsf{N}_1 \to \mathsf{N}_2$$
$$F_{\mathsf{V}} : \mathsf{V}_1 \to \mathsf{N}_2 + \mathsf{V}_2$$

satisfying the following conditions:

$$F_{\mathsf{N}} \mathbin{;} \mathsf{lab}_2 = \mathsf{lab}_1$$
$$F_{\mathsf{N}} \mathbin{;} \mathsf{suc}_2 = \mathsf{suc}_1 \mathbin{;} \mathsf{List}((F_{\mathsf{N}} + F_{\mathsf{V}}) \mathbin{;} \mu_{(\mathsf{N}_2+)})$$

where $\mu_{(\mathsf{N}+)} : \forall X \,.\, (\mathsf{N} + (\mathsf{N} + X)) \to (\mathsf{N} + X)$ is the canonical flattening function for nested alternatives with $\mathsf{N}$. □

This time, we are dealing with a parameterised monad, namely $(\mathsf{N}+)$, which maps any $X$ to $\mathsf{N}+X$, where the parameter $\mathsf{N}$ is instantiated with the respective carrier of that sort. Composition of the $\mathsf{V}$ components of $F : T_1 \to T_2$ and $G : T_2 \to T_3$ is defined accordingly:

$$(F \mathbin{;} G)_{\mathsf{V}} = F_{\mathsf{V}} \mathbin{;} (G_{\mathsf{N}} + G_{\mathsf{V}}) \mathbin{;} \mu_{(\mathsf{N}_3+)}$$

Again, the resulting category is well-defined.✓

### 6.3 Generalised Coalgebra Morphisms

For obtaining the general shape of such "monadic coalgebra morphisms", inspection of the signatures shows that each of the functors underlying these kinds of coalgebras not only contains a primitive monad (the term monad for attributed graphs, and an "alternative monad" for term graphs), but even can be factored over a monad on the relevant product category.

Since the signature $\mathsf{sigAG}_\Sigma$ has three sorts, the underlying category is the triple product $\mathit{Set} \times \mathit{Set} \times \mathit{Set}$, with triples of sets as objects.

The coalgebra functor for $\mathsf{sigAG}_\Sigma$ is then

$$\mathcal{G}_{\mathsf{sigAG}_\Sigma}\ (N, E, V) = (\mathbb{1}, (N \times N \times \mathcal{T}_\Sigma\ V), \mathbb{1})\ ,$$

mapping node and edge set to the terminal object $\mathbb{1}$ since no operations take nodes or edges as arguments; since there are three operations taking edges as arguments, $\mathcal{G}_{\mathsf{sigAG}_\Sigma}$ produces as "edge component" of its result type the Cartesian product $N \times N \times \mathcal{T}_\Sigma\ V$ consisting of the target sets of the three operations.

We can decompose this as $\mathcal{G}_{\mathsf{sigAG}_\Sigma} = \mathcal{F}_{\mathsf{sigAG}_\Sigma} \circ \mathcal{M}_{\mathsf{sigAG}_\Sigma}$, where:

$$\begin{aligned}\mathcal{M}_{\mathsf{sigAG}_\Sigma}\ (N, E, V) &= (N, E, \mathcal{T}_\Sigma\ V)\\ \mathcal{F}_{\mathsf{sigAG}_\Sigma}\ (N, E, T) &= (\mathbb{1}, (N \times N \times T), \mathbb{1})\end{aligned}$$

Since $\mathcal{M}_{\mathsf{sigAG}_\Sigma}$ is the product of twice the identity monad with the term monad $\mathcal{T}_\Sigma$, it is obviously a monad.✓

Analogously, the coalgebra functor for $\mathsf{sigTG}$ is

$$\mathcal{G}_{\mathsf{sigTG}}\ (N, V) = (L \times \mathsf{List}\ (N + V), \mathbb{1})\ ,$$

mapping the variable set to the terminal object $\mathbb{1}$ since no operations take variables as arguments. We can decompose this as $\mathcal{G}_{\mathsf{sigTG}} = \mathcal{F}_{\mathsf{sigTG}} \circ \mathcal{M}_{\mathsf{sigTG}}$, where:

$$\begin{aligned}\mathcal{M}_{\mathsf{sigTG}}\ (N, V) &= (N, (N + V))\\ \mathcal{F}_{\mathsf{sigTG}}\ (N, S) &= (L \times \mathsf{List}\ S, \mathbb{1})\end{aligned}$$

It is straightforward to prove that $\mathcal{M}_{\mathsf{sigTG}}$ is a monad constructed as a "dependent product monad".✓

In general, we define:

**Definition 6.3** Given a monad $\mathcal{M}$ and an endofunctor $\mathcal{F}$ over a category $\mathcal{C}$, an *$\mathcal{M}$-$\mathcal{F}$-coalgebra* is a coalgebra over the functor $\mathcal{F} \circ \mathcal{M}$, that is, a pair $(A, \mathsf{op}_A)$ consisting of

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

A *raw $\mathcal{M}$-$\mathcal{F}$-coalgebra homomorphism* from $(A, \mathsf{op}_A)$ to $(B, \mathsf{op}_B)$ is a morphism from $A$ to $B$ in the Kleisli category of $\mathcal{M}$; raw morphism composition is Kleisli composition. □

One might expect that we obtain just coalgebras over the Kleisli category of $\mathcal{M}$. However, the following complications hold:

- $\mathcal{F}$ does in general not give rise to a functor over the Kleisli category of $\mathcal{M}$.
- If a natural transformation from $\mathcal{F} \circ \mathcal{M}$ to $\mathcal{M} \circ \mathcal{F}$ exists, an endofunctor on the Kleisli category that coincides with $\mathcal{F}$ on objects can be constructed — however, no such a natural transformation exists for $\mathsf{sigAG}_\Sigma$.✓
- Constructing an endofunctor on $\mathcal{C}$ from an endofunctor on the Kleisli category would require transformations to "extract from the monad $\mathcal{M}$" which can be natural neither on $\mathcal{C}$ nor on the Kleisli category.✓

In addition, not all raw $\mathcal{M}_{\mathsf{sigAG}_\Sigma}$-$\mathcal{F}_{\mathsf{sigAG}_\Sigma}$-coalgebra homomorphisms satisfy the conditions we listed above for monadic $\mathsf{sigAG}_\Sigma$ coalgebra homomorphisms — we need to identify an appropriate subcategory of the Kleisli category.

From the material we have, we can easily construct the following two morphisms:

$$\begin{array}{rcl} f \mathbin{;} \mathcal{M}\ \mathsf{op}_B & : & A \to \mathcal{M}\mathcal{F}\mathcal{M}B \\ \mathsf{op}_A \mathbin{;} \mathcal{F}\mathcal{M}f & : & A \to \mathcal{F}\mathcal{M}\mathcal{M}B \end{array}$$

"Obviously", we can complete this to a commutativity condition using a natural transformation constructed from the return $\eta$ and the join $\mu$ transformations of the monad $\mathcal{M}$, namely:

$$\mathcal{F}\mu \mathbin{;} \eta \ : \ \mathcal{F} \circ \mathcal{M} \circ \mathcal{M} \Rightarrow \mathcal{M} \circ \mathcal{F} \circ \mathcal{M}$$

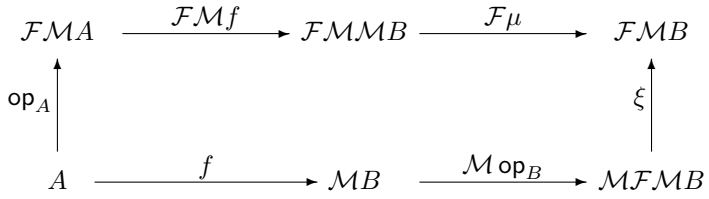
For the category $\mathsf{AG}_\Sigma$ of Def. 6.1, this condition is unfortunately only satisfied by morphisms where $F_\mathsf{V}$ only renames variables✓, which defeats our intentions. This problem is actually not even due to the choice of $\mathcal{F}\mu \mathbin{;} \eta$, but to the choice of direction, since it arises for every natural transformation from $\mathcal{F} \circ \mathcal{M} \circ \mathcal{M}$ to $\mathcal{M} \circ \mathcal{F} \circ \mathcal{M}$. Therefore, using distribution transformations from $\mathcal{F} \circ \mathcal{M}$ to $\mathcal{M} \circ \mathcal{F}$ as used by Pardo [Par98] is not an option either.

We found that natural transformations from $\mathcal{M} \circ \mathcal{F} \circ \mathcal{M}$ to $\mathcal{F} \circ \mathcal{M}$ "work" when combined with a join on the other side:

**Definition 6.4** For an endofunctor $\mathcal{F}$ and a monad $(\mathcal{M}, \eta, \mu)$ on $\mathcal{C}$, an $\mathcal{M}$-$\mathcal{F}$-*distrjoin* transformation is a natural transformation $\xi : \mathcal{M} \circ \mathcal{F} \circ \mathcal{M} \Rightarrow \mathcal{F} \circ \mathcal{M}$ for which the following properties hold:

- $\eta \mathbin{;} \xi = \mathbb{I}$
- $\mu \mathbin{;} \xi = \mathcal{M}\xi \mathbin{;} \xi$
- $\mathcal{M}\mathcal{F}\mu \mathbin{;} \xi = \xi \mathbin{;} \mathcal{F}\mu$ □

**Definition 6.5** Given an endofunctor $\mathcal{F}$ and a monad $(\mathcal{M}, \eta, \mu)$ on $\mathcal{C}$, and also an $\mathcal{M}$-$\mathcal{F}$-distrjoin transformation $\xi$, an $\mathcal{M}$-$\mathcal{F}$-*coalgebra homomorphism* from $(A, \mathsf{op}_A)$ to $(B, \mathsf{op}_B)$ is a morphism $f : A \to \mathcal{M}B$ making the following diagram commute:




**Theorem 6.6** $\mathcal{M}$-$\mathcal{F}$-coalgebras with such $\mathcal{M}$-$\mathcal{F}$-coalgebra homomorphisms form a category.✓ □

The instantiations of this for edge-attributed graphs ($\mathsf{sigAG}_\Sigma$) and for term graphs ($\mathsf{sigTG}$) are appropriate:

**Theorem 6.7** The $\mathcal{M}_{\mathsf{sigAG}_\Sigma}$-$\mathcal{F}_{\mathsf{sigAG}_\Sigma}$-coalgebra category is equivalent to the category $\mathsf{AG}_\Sigma$ of Def. 6.1.✓ □

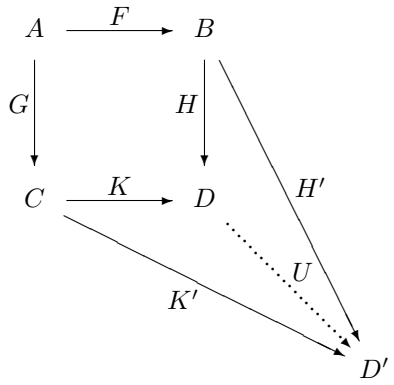
**Theorem 6.8** The $\mathcal{M}_{\mathsf{sigTG}}$-$\mathcal{F}_{\mathsf{sigTG}}$-coalgebra category is equivalent to the category $\mathsf{TG}$ of Def. 6.2.✓ □

### 6.4 Pushouts of Monadic Coalgebra Morphisms

It is well-known that the Kleisli category over the term monad $\mathcal{T}$ does not have all pushouts, and that existence of pushouts essentially corresponds to unifiability.

Therefore we cannot expect the category of monadic $\mathsf{sigAG}_\Sigma$ coalgebras to have all pushouts. Nevertheless, since pushouts are the key ingredient of the categoric approach to graph transformation, an interesting question is whether pushouts in the Kleisli category of $\mathcal{M}$ give rise to pushouts in the $\mathcal{M}$-$\mathcal{F}$-coalgebra category. It is easy to see that this is not the case for term graphs; however, it does hold for symbolically edge-attributed graphs, and in this section we explore the question how to prove this from the point of view of $\mathcal{M}$-$\mathcal{F}$-coalgebra categories without considering decompositions of $\mathcal{M}$ and $\mathcal{F}$.

A pushout for a span $B \xleftarrow{F} A \xrightarrow{G} C$ is a completion $B \xrightarrow{H} D \xleftarrow{K} C$ to a commuting square that is "minimal" in the sense that every other candidate completion factors uniquely over it (via $U$).



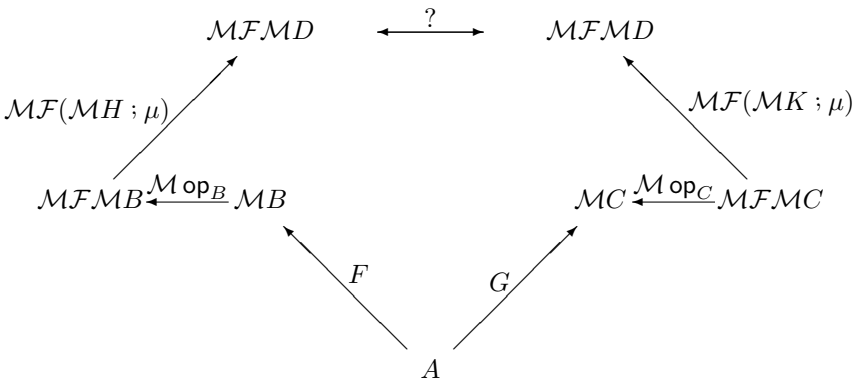

Assuming such a pushout in the Kleisli category of $\mathcal{M}$, for constructing the operation $\mathsf{op}_D$ of the target coalgebra we need to choose appropriate $H'$ and $K'$ such that the $U$ can be used to construct $\mathsf{op}_D$ and prove its pushout property in the category of $\mathcal{M}$-$\mathcal{F}$-coalgebras.

Without assuming additional (natural) transformations, we can only choose the following:

$$H' = \mathsf{op}_B \mathbin{;} \mathcal{F}(\mathcal{M}H \mathbin{;} \mu) \mathbin{;} \eta$$
$$K' = \mathsf{op}_C \mathbin{;} \mathcal{F}(\mathcal{M}K \mathbin{;} \mu) \mathbin{;} \eta$$

However, commutativity $F \fatsemi H' = G \fatsemi K'$ in the Kleisli category can only be shown assuming additional (natural) transformations and/or laws, which however are

not available for the setting of $\mathsf{sigAG}_\Sigma$ — there, this commutativity does not hold.✓ The essential reason for this is that $\mathcal{F}$ maps $V$ to $\mathbb{1}$, while $V$ is also the only component that has a non-trivial monad. Commutativity fails for the $V$ components due to the fact that $\mathsf{op}_B$ and $\mathsf{op}_C$ map these to $\mathbb{1}$, while $\mathcal{M}$ for the $V$ components is the term monad $\mathcal{T}_\Sigma$:




Due to this property of the operators in $\mathsf{sigAG}_\Sigma$, commutativity will actually fail for any definition of $H'$ of the shape "$H' = \mathsf{op}_B\,;\ldots$". We solve this problem in the next section by restriction to more specialised versions of $\mathcal{M}$ and $\mathcal{F}$, which allows us to "patch" $H'$ and $K'$ so as to avoid this conflict.

## 7 Monadic Product Coalgebras

We now specialise the $\mathcal{M}$-$\mathcal{F}$-coalgebras of Sect. 6.3 in a way that still generalises the setup for symbolically edge-attributed graphs there, while also allowing a pushout construction.

The most general shape we have been able to identify for this are "monadic product coalgebras" over a product category $\mathcal{C}_1 \times \mathcal{C}_2$, defined in the following setting (which we will assume for the remainder of this section): Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two categories; let $\mathcal{M}$ be a monad on $\mathcal{C}_2$, and $\mathcal{F}$ a functor 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 7.1** 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

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

Morphism composition is composition of the corresponding product category. □

This morphism composition is well-defined✓, and induces a category ✓.

Now let $\mathcal{M}_0$ be the product monad of the identity monad on $\mathcal{C}_1$ and $\mathcal{M}$, and define $\mathcal{F}_0$ as endofunctor on $\mathcal{C}_1 \times \mathcal{C}_2$ by:

$$\mathcal{F}_0(X_1, X_2) = (\mathcal{F}(X_1, X_2), \mathbb{1})$$

With these definitions, the $\mathcal{M}_0$-$\mathcal{F}_0$-distrjoin transformation (see Def. 6.4) has identities of $\mathcal{C}_1$ and terminal morphisms of $\mathcal{C}_2$ as its two components✓, which allows us to identify monadic product coalgebras as a special case of the monadic coalgebras of Sect. 6.3:

**Theorem 7.2** The category of $\mathcal{M}$-$\mathcal{F}$-product-coalgebra homomorphisms is equivalent to the category of $\mathcal{M}_0$-$\mathcal{F}_0$-coalgebra homomorphisms.✓ □

In addition, the more fine-grained structure of monadic product coalgebras allows us to circumvent the problems we encountered in Sect. 6.4.

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

**Theorem 7.3** Let a span $B \xleftarrow{F} A \xrightarrow{G} C$ of $\mathcal{M}$-$\mathcal{F}$-product-coalgebra homomorphisms be given, and a cospan $(B_1, B_2) \xrightarrow{H} (D_1, D_2) \xleftarrow{K} (C_1, C_2)$ in $\mathbb{K}$ 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.✓

**Proof sketch:** The first step is the construction of a cospan

$$(B_1, B_2) \quad \xrightarrow{H'} \quad D' \quad \xleftarrow{K'} \quad (C_1, C_2)$$

in $\mathbb{K}$ such that the first component of the universal morphism $U : (D_1, D_2) \to D'$ from the $\mathbb{K}$ pushout can be used as $\mathsf{op}_D$.

The first constituent of $D'$ therefore must be the target of $\mathsf{op}_D$, so we define:

$$\begin{aligned} D' &= (\quad\quad \mathcal{F}(D_1, \mathcal{M}\ D_2) \quad\quad , D_2) \\ H' &= (\ \mathsf{op}_B \mathbin{;} \mathcal{F}(H_1, \mathcal{M}\ H_2 \mathbin{;} \mu)\ , H_2) \\ K' &= (\ \mathsf{op}_C \mathbin{;} \mathcal{F}(K_1, \mathcal{M}\ K_2 \mathbin{;} \mu)\ , K_2) \end{aligned}$$

The second constituent of $D'$ is inherited from $(D_1, D_2)$, which allows us to use the universality of the original $\mathbb{K}$ pushout when proving universality of the resulting $\mathcal{M}$-$\mathcal{F}$-product-coalgebra pushout. □

Together with the two equivalences of categories of Theorems 7.2 and 6.7, pushouts for edge-attributed graphs essentially reduce to unification for their variable components (if we choose an underlying category that has pushouts, such as *Set*):

**Corollary 7.4** A span $B \xleftarrow{F} A \xrightarrow{G} C$ in the $\mathcal{M}_{\mathsf{sigAG}}$-$\mathcal{F}_{\mathsf{sigAG}}$-coalgebra category over *Set* for edge-attributed graphs (as $\mathsf{sigAG}_\Sigma$ structures) has a pushout if $F_\mathsf{V}$ and $G_\mathsf{V}$, as substitutions, have a pushout.✓ □

## 8 Conclusion and Outlook

We have shown how the additional flexibility of coalgebraic signatures enables us to integrate label types and symbolic attribute types into graph structure signatures, and then modified the coalgebraic homomorphism concept via integration of Kleisli arrows to achieve the flexibility necessary to allow substitutions as part of our morphisms. We showed that several seemingly plausible formalisations for this do not model the intended applications, and arrived at a simple factoring setup (Sect. 6.3) that additionally encompasses a natural formalisation of term graphs. For symbolically attributed graphs fitting into the pattern of monadic product coalgebras (Sect. 7), we showed that pushouts can be obtained where the substitution components of their homomorphisms are unifiable.

Without the support of our mechanised formalisation in Agda, the mentioned failures of inappropriate formalisations, and also the successful proofs reported in the paper would have been extremely hard to arrive at with comparable confidence.

Since pushouts do not necessarily exist in Kleisli categories, an important question is whether general results for appropriate classes of restricted monomorphisms can be obtained. In this context, the "guarded monads" of Ghani *et al.* [GLDM05] might be useful, since a guarded monad essentially can be represented as $\mathsf{Id} + \mathcal{N}$ for some functor $\mathcal{N}$, and the identity component can be used to lift monomorphisms from the base category into the Kleisli category.

The ultimate goal of this work is a fully verified implementation of monadic coalgebra transformation that can be instantiated in particular for the transformation of symbolically attributed graph structures.

*Acknowledgements.* I am grateful to the anonymous referees for their constructive comments.

## References

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

[BM04] Bidoit, M., Mosses, P. D.: CASL User Manual, LNCS (IFIP Series), vol. 2900. Springer (2004). With chapters by T. Mossakowski, D. Sannella, and A. Tarlecki

[Cap11] Capretta, V.: Coalgebras in functional programming and type theory. Theoretical Computer Science 412, 5006–5024 (2011)

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

[CEKR02] Corradini, A., Ehrig, H., Kreowski, H., Rozenberg, G. (eds.) ICGT 2002, LNCS, vol. 2505 (2002)

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

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

[EPT04] Ehrig, H., Prange, U., Taentzer, G.: Fundamental Theory for Typed Attributed Graph Transformation. In [PPBE04], pp. 161–177

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

[GLDM05] Ghani, N., Lüth, C., De Marchi, F.: Monads of Coalgebras: Rational Terms and Term Graphs. Mathematical Structures in Computer Science 15, 433–451 (2005)

[HP02] Habel, A., Plump, D.: Relabelling in Graph Transformation. In [CEKR02], pp. 135–147

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

[HKT02] Heckel, R., Küster, J. M., Taentzer, G.: Confluence of Typed Attributed Graph Transformation Systems. In [CEKR02], pp. 161–176

[Kah11] Kahl, W.: Dependently-Typed Formalisation of Relation-Algebraic Abstractions. In de Swart, H. (ed.) RAMiCS 2011, LNCS, vol. 6663, pp. 230–247. Springer (2011)

[Kah14] Kahl, W.: Relation-Algebraic Theories in Agda — RATH-Agda-2.0.1. Mechanically checked Agda theories available for download, with 456 pages literate document output. http://RelMiCS.McMaster.ca/RATH-Agda/. (2014)

[KK08] König, B., Kozioura, V.: Towards the Verification of Attributed Graph Transformation Systems. In Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008, LNCS, vol. 5214, pp. 305–320 (2008)

[KH02] Kurz, A., Hennicker, R.: On Institutions for Modular Coalgebraic Specifications. Theoretical Computer Science 280, 69–103 (2002)

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

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

[Nor07] Norell, U.: Towards a Practical Programming Language Based on Dependent Type Theory. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology (2007)

[OL10] Orejas, F., Lambers, L.: Delaying Constraint Solving in Symbolic Graph Transformation. In Ehrig, H., Rensink, A., Rozenberg, G., Schürr, A. (eds.) ICGT 2010, LNCS, vol. 6372, pp. 43–58 (2010)

[Par98] Pardo, A.: Monadic Corecursion — Definition, Fusion Laws and Applications. ENTCS 11, (1998)

[PPBE04] Parisi-Presicce, F., Bottoni, P., Engels, G. (eds.) ICGT 2004, LNCS, vol. 3256 (2004)

[PS04] Plump, D., Steinert, S.: Towards Graph Programs for Graph Algorithms. In [PPBE04], pp. 128–143

[Plu09] Plump, D.: The Graph Programming Language GP. In: CAI 2009, LNCS, vol. 5725, pp. 99–122 (2009)

[PZ01] Poll, E., Zwanenburg, J.: From Algebras and Coalgebras to Dialgebras. ENTCS 44, 289–307 (2001)

[RFS08] Rebout, M., Féraud, L., Soloviev, S.: A Unified Categorical Approach for Attributed Graph Rewriting In Hirsch, E., Razborov, A., Semenov, A., Slissenko, A. (eds.) CSR 2008, LNCS, vol. 5010, pp. 398–409. Springer (2008)

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

[Rut00] Rutten, J. J.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249, 3–80 (2000)