A General Attribution Concept for Models in $\mathcal{M}$-Adhesive Transformation Systems

Ulrike Golas

Konrad-Zuse-Zentrum für Informationstechnik Berlin, Germany
golas@zib.de

Abstract. Attributes are an important concept for modeling data in practical applications. Up to now there is no adequate way to define attributes for different kinds of models used in $\mathcal{M}$-adhesive transformation systems, which are a special kind of graph transformation system based on $\mathcal{M}$-adhesive categories. Especially a proper representation and definition of attributes and their values as well as a suitable handling of the data does not fit well with other graph transformation formalisms.

In this paper, we propose a new method to define attributes in a natural, but still formally precise and widely applicable way. We define a new kind of adhesive category, called $\mathcal{W}$-adhesive, that can be used for transformations of attributes, while the underlying models are still $\mathcal{M}$-adhesive ones. As a result, attributed models can be used as they are intended to be, but with a formal background and proven well-behavior.

1 Introduction

Graph transformation is a well-known formalism for the rule-based derivation of graphs [1,2]. For different application areas, different graph types are necessary to express all the properties of the underlying models. An underlying category theoretical framework provides the possibility to once and for all prove different properties of graph transformation, like the local Church-Rosser property or local confluence, on the categorical level and then transfer it by instantiation to the different graphs. $\mathcal{M}$-adhesive categories [3] and their variants like adhesive [4] and weak adhesive HLR [5] categories form such a framework for graph transformations and their analysis. They are based on a distinguished morphism class $\mathcal{M}$, which is used for the rule morphisms and has to satisfy certain properties, and a special van Kampen (VK) property describing the compatibility of gluings and restrictions, i.e. pushouts and pullbacks in categorical terms. Based on this van Kampen property, many results can be proven that hold for all kinds of graphs.

Attributes play an important role for modeling, for example in object-oriented system models or network graphs. But up to now, no general method for the definition of attributes in arbitrary models has been found which represents attributes as understood in object-oriented systems. In theoretical contributions, often an algebraic approach combined with graphs is used which has disadvantages like allowing multiple attribute values for one attribute or integrating the complete data values into the graph structure leading to infinite graphs. While

H. Ehrig et al.(Eds.): ICGT 2012, LNCS 7562, pp. 187–202, 2012.

in other contexts, like conflict resolution of diverging transformations in model versioning [6], multiple attribute values may be useful, they are unexpected and lead to strange behavior in the general modeling of object-oriented systems.

The idea of this contribution is a separation of concerns, i.e. the separation of the graph structure and their attribution and data. The goal is to have a formalism that allows users to use attributes as in object-oriented models, but with a formal background. It should be applicable to all kinds of graphs and graph-like structures and compatible with well-known graph transformation approaches. We will not achieve a simple attribution concept – which does not seem to exist in a formal and wide-ranging way – but a theoretically well-founded one, which can be applied intuitively without the need to take care of unexpected side effects as necessary for current approaches. The formalization leads to the notion of $\mathcal{W}$-adhesive categories, where the main idea is to restrict the VK property to those squares that actually appear in transformations.

This paper is organized as follows. In Section 2, we motivate our work by analyzing how attributes should work, how they are defined in current approaches and describe the general idea of our new attribution concept. This is defined formally in Section 3. In Section 4, we define transformations of attributed objects and show that the attribution concept does not lead to $\mathcal{M}$-adhesive categories. Instead, $\mathcal{W}$-adhesive categories are defined in Section 5, where we show exemplarily the Local Church-Rosser Theorem to illustrate that this category is suitable for graph transformation. A conclusion and future work are given in Section 6.

We assume the reader to be familiar with graph transformation in the double pushout approach and the foundations of $\mathcal{M}$-adhesive transformation systems as, for example, introduced in [5]. The proofs of the theorems and facts, as far as they are not directly included in this paper, can be found in [7].

2 Motivation and Related Work

When working with attributes, most people expect attributes to behave like in object-oriented models. This means in particular that

- a model element has exactly one value for each attribute and
- model elements of different types may have the same attribute.

Unfortunately, this is not true for typed attributed graphs [5]. Typed attributed graphs are based on so-called E-graphs, which are graphs consisting of different node and edge types. The actual attribution of a node is done by an attribute edge of a certain type, representing this attribute's name, pointing from this node to the data value. Thus, also multiple attribute edges of the same type from a node form a valid graph, which means that the node has different values for the same attribute. Moreover, the attribute edges form a set which means that for different types we are not allowed to use the same attribute name. This situation is shown in Fig. 1. In the left, we see a class diagram-like graph modeling a `cat` with `name = "Greebo"` and a `dog` with `name = "Gaspode"`, which is what we expect to see. In the middle, the corresponding type graph is shown. Since `Cats`

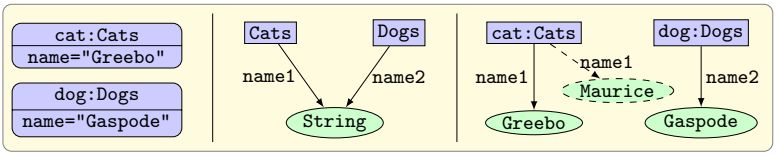


Fig. 1. Attribution in typed attributed graphs

and `Dogs` are different types, the attribute edge names have to be different, i.e. something like `name1` and `name2`. In the right, the typed attributed graph for our model is shown, where the typing of each element is denoted after the colon. While we would not expect a second name attribute for an object in the left representation, it could occur on the right, indicated by the dotted elements. To avoid this we had to define constraints forbidding the occurrence of this kind of double edges. Especially, we have to include application conditions for each rule to ensure that attribute edges can only be created if they do not already exist for the considered node. Another difficulty with this approach is that the data values are completely stored in the graph and are not only implicitly there, which means that the graph becomes infinite in general.

Symbolic graphs [8] are graphs labeled with variables and combined with formulas over these variables and a data algebra. Due to this concept, they allow for a separation of the graph and data part, such that the graphs themselves are not infinite only because of their data. In general, a symbolic graph represents a set of attributed graphs satisfying the formulas and allows for a logical reasoning about data. Nevertheless, the underlying graph structure is an E-graph, like for typed attributed graphs, and has the same behavior allowing multiple values for the same attribute, but needing different attribution names for different types.

Transformations of partially labelled graphs [9], i.e. graphs where nodes and edges may be labelled by (disjoint) label alphabets, can be treated somehow similar to attributes. But the overall setting in this approach is different from the standard version of double-pushout transformations. While we want to have injective rule morphisms and arbitrary matches, in [9] only the left rule morphism has to be injective, but in addition the match.

Other approaches also leave the context of $\mathcal{M}$-adhesive transformation systems like coding the graphs as algebras [10] or combining graphs with type theory for attribution [11], which needs a different transformation approach, since the attribution transformation is done via pullbacks instead of pushouts.

Basic Idea

For rule-based transformations, attribute values need to be preserved or refined. Note that actually changing attributes themselves would be a transformation in the meta model and is not our concern here. In addition to the creation or deletion of complete nodes including all attributes, our rules should be able to:

1. keep an attribute value, i.e. leave it unchanged (top of Fig. 2), or
2. change an attribute value, i.e. assign a new value to it (bottom of Fig. 2).

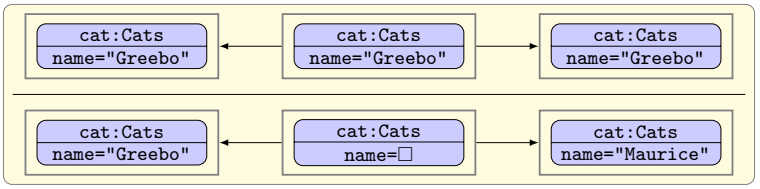


Fig. 2. Typical attribute behavior in rules

To allow this, we need "undefined" attributes, which are denoted by □. They should only appear in the interface of the rule and the corresponding intermediate graph models. For the morphisms used in rules, only certain mappings are allowed, such that all undefined attributes are concretized by a data value, and all other attribute values are identically mapped.

3 Attribution of Models

In this section, we define the category **AttC** of attributed objects with an underlying category **C**. To reach this goal, we have to formalize the ideas of the previous section. The key idea is to separate the (graphical) model and the data values used for attribution.

Attribution Values
Consider a finite set TYPES of available data types and for each $type \in \texttt{TYPES}$ we have a set A_{type} containing the data values. They may come, for example, from the sorts of a signature and their respective carrier sets in an algebra, an abstract data type, or a programming language. Moreover, not only concrete values (like all integers) but also abstract values (like all terms over variables of a certain signature [12]) can be used for attribution.

Definition 1 (Attribution values). *Given the data types* TYPES*,* attribution values $A = \dot{\cup}_{type \in \texttt{TYPES}} A_{type}$ *are defined by a set* A_{type} *of data values for each* $type \in \texttt{TYPES}$.

Definition 2 (Attribution value morphism). *Consider attribution values* A_1 *and* A_2*, an attribution value morphism* $a : A_1 \to A_2$ *is defined by a family of functions* $(a_{type} : A_{1,type} \to A_{2,type})_{type \in \texttt{TYPES}}$ *such that* $a(x) = a_{type}(x)$ *for all* $x \in A_{1,type}$.

Example 1. For our running example, we only need one data type of strings, which means that $\texttt{TYPES} = \{string\}$. For models, we use as attribution values A_{string} the set of all strings. Moreover, we need attribution values for rules, where we use the set $T_{string}(\mathcal{V})$ of terms over variables $\mathcal{V} = \{*_1, *_2, *_3\}$, where certain standard string operations are defined. For an attribution value morphism $a : T(\mathcal{V}) \to A$, we use a variable assignment $\alpha : \mathcal{V} \to A$ such that a is the evaluation of terms according to α.

Attribution of Models
Consider different types of elements, which all should be attributed the same way, for example the types of a type graph. For each of these attribution types, we define a set of attribute names with a certain type. For readability, we do not allow the same attribute name with different type for one attribution type.

Definition 3 (Typed attribute). *Let $\mathcal{K}$ be a finite set whose elements are called attribution types and* Voc *a set (or vocabulary) of available attribute names. The mapping $Atts : \mathcal{K} \to \mathcal{P}(\text{Voc} \times \texttt{TYPES})$ defines a set of* typed attributes *for each $k \in \mathcal{K}$, if (s,t), $(s,t') \in Atts(k)$ for some $k \in \mathcal{K}$ implies that $t = t'$. The set $att_{name}(k)$ is the projection of $Atts(k)$ to its first component.*

For the attribution of the elements of a certain model, these have to be grouped into the attribution types $\mathcal{K}$. This is done by a so-called $\mathcal{K}$-functor. Note that the sets are not necessarily disjoint, which means that an element may have multiple attribution types. This can be used, for example, to express inheritance.

Definition 4 ($\mathcal{K}$-functor). *For a category $\mathbf{C}$, a pushout-preserving functor $F : \mathbf{C} \to \mathbf{Sets}^{\mathcal{K}}$ is said to be a* $\mathcal{K}$-functor*, where $\mathbf{Sets}^{\mathcal{K}}$ is the $|\mathcal{K}|$-fold product of* **Sets**. *With $F_k : \mathbf{C} \to \mathbf{Sets}$ we denote the (also pushout-preserving) functor as projection of F to its k-th component.*

Using all the above definitions, we can now attribute the models. The $\mathcal{K}$-functor combined with *Atts* defines which attributes are mapped to which model element. We need an additional mapping of these attribute names to actual data values consistent with their defined type. This mapping may be partial, because we allow that certain attributes are not set. Note that for partial functions f and f', with $f(a) = f'(a)$ we denote that both $f(a)$ and $f'(a)$ are defined and equal, or both are undefined. Moreover, we write $f \leq f'$, if $f(d) = f'(d)$ for all $d \in Dom(f)$.

Definition 5 (Attributed object). *Given a category $\mathbf{C}$ and a $\mathcal{K}$-functor $F : \mathbf{C} \to \mathbf{Sets}^{\mathcal{K}}$ then $AO = (C, A, att)$ is an* attributed object *if $C \in \mathbf{C}$ with attribution values A and $att = (att_k : F_k(C) \times att_{name}(k) \nrightarrow A)_{k \in \mathcal{K}}$ is a family of partial attribution functions such that for all $att_k(o,n) \in A_{type}$ we have that $(n, type) \in Atts(k)$.*

Example 2. We enrich our running example with humans. We define the attribution types $\mathcal{K} = \{pet, hum\}$ of pets and humans, where pets have a name, i.e. $Atts(pet) = \{(\texttt{name}, String)\}$, while humans have a first and a last name, i.e. $Atts(hum) = \{(\texttt{fname}, String), (\texttt{lname}, String)\}$. Note that we are free to define also an attribute `name` for humans, which even may have a different type than *String* (if other types were available for our example). In this example, we only use node attribution, but of course edge attribution would also be possible – we only had to add an attribution type for the edges.

For the category $\mathbf{C}$ we choose the category of typed graphs using the type graph TG shown in the left of Fig. 3 defining cats, dogs, humans, and owns

relations between humans and pets. An example typed graph G is depicted in the right of Fig. 3 defining two cats, one dog, and one human, which owns one of the cats. Note that actually we would need two different owns relations due to the uniqueness of edges in the type graph, but we omit this here. While we allow the same attributes for different elements within our attribution concept, we cannot solve the same problem for types in the underlying typed graphs. To model one owns relation for all pets we could implement an inheritance concept (see [5]) and extract a class pet.

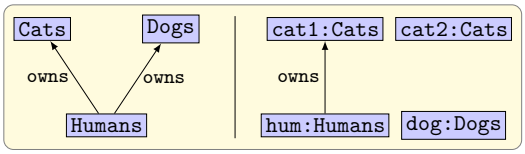


Fig. 3. Type graph TG and typed graph G

The $\mathcal{K}$-functor $F : \mathbf{Graphs_{TG}} \to \mathbf{Sets}^{\mathcal{K}}$ includes each node of type `Cats` or `Dogs` into the *pet*-set, and each human into the *hum*-set. This means that $F(G) = (\{n \mid type(n) \in \{\texttt{Cats}, \texttt{Dogs}\}\}, \{n \mid type(n) = \texttt{Humans}\})$, while morphisms are mapped to their corresponding components. For the example graph G in Fig. 3 this means that $F(G) = (\{\texttt{cat1}, \texttt{cat2}, \texttt{dog}\}, \{\texttt{hum}\})$.

An attributed object $AO' = (G, A, att')$ is shown in Fig. 4, where the underlying typed graph G is already given in Fig. 3. The attribution is defined by $att'_{pet}(\texttt{cat1}, \texttt{name}) = \texttt{"Greebo"}$, $att'_{pet}(\texttt{cat2}, \texttt{name}) = \texttt{"Maurice"}$, $att'_{pet}(\texttt{dog}, \texttt{name}) = \texttt{"Gaspode"}$, $att'_{hum}(\texttt{hum}, \texttt{fname}) = \texttt{"Nanny"}$, and $att'_{hum}(\texttt{hum}, \texttt{lname}) = \texttt{"Ogg"}$.

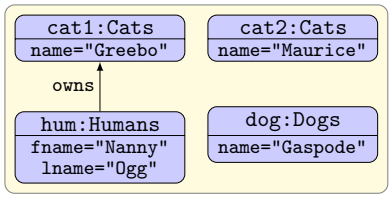
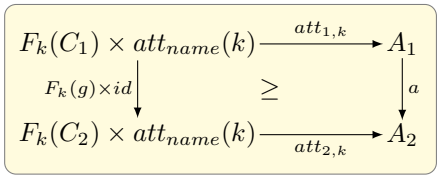


Fig. 4. An attributed graph

Morphisms

To express relations between attributed objects and apply graph transformation, we need to define attributed morphisms. We combine a valid morphism in the underlying category with an attribution value morphism to obtain attribution morphisms, where we allow to concretize undefined attribute values.

Definition 6 (Attributed morphism). *Consider a set $\mathcal{A}$ of available attribution value morphisms closed under composition.*



Given attributed objects $AO_1 = (C_1, A_1, att_1)$ and $AO_2 = (C_2, A_2, att_2)$, an attributed morphism $f : AO_1 \to AO_2$ *is a pair $f = (g, a)$ with $g : C_1 \to C_2 \in \mathbf{C}$ and $a : A_1 \to A_2 \in \mathcal{A}$ such that $a \circ att_{1,k} \leq att_{2,k} \circ (F_k(g) \times id)$.*

We use the following convention: When given an attributed morphism f_i, we denote its components with g_i and a_i, i.e. $f_i = (g_i, a_i)$. Note that the set $\mathcal{A}$ restricts the available attribution value morphisms. This is useful if these

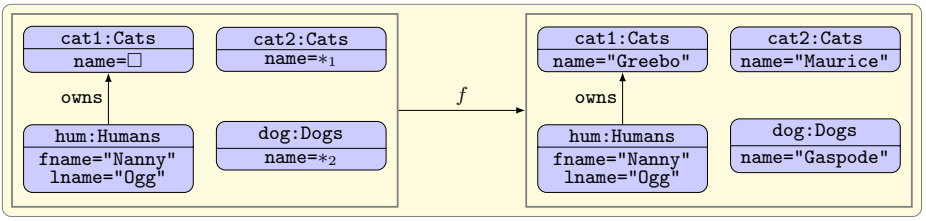


Fig. 5. An attributed morphism

morphisms shall preserve certain structure. In our example, we only want to use attribution value morphisms that stem from an evaluation based on a variable assignment.

Example 3. In Fig. 5, an attributed morphism $f = (g, a)$ from an attributed graph $AO = (G, T^{\mathcal{V}}(A), att)$ to $AO' = (G, A, att')$ is shown. The attribution of G in AO is defined by $att_{pet}(\texttt{cat2}, \texttt{name}) = *_1$, $att_{pet}(\texttt{dog}, \texttt{name}) = *_2$, $att_{hum}(\texttt{hum}, \texttt{fname}) = \texttt{"Nanny"}$, and $att_{hum}(\texttt{hum}, \texttt{lname}) = \texttt{"Ogg"}$, and $att_{pet}(\texttt{cat1}, \texttt{name})$ is undefined. AO' is already known from Fig. 4. g is the identical morphism on the graph G, while a is the evaluation with $\alpha(*_1) = \texttt{"Maurice"}$ and $\alpha(*_2) = \texttt{"Gaspode"}$.

To show that f is a valid attributed morphism we have to verify its properties for all $(o, n) \in Dom(a \circ att_k)$ with $k \in \{pet, hum\}$, which obviously holds.

Now attributed objects and attributed morphisms form a category.

Definition 7 (Category AttC). *Given a category* $\mathbf{C}$*, a set* $\mathcal{A}$ *of attribution element morphisms, a finite set of attribution types* $\mathcal{K}$*, a* $\mathcal{K}$*-functor* F*, and a mapping Atts as above, then attributed objects and attributed morphisms, together with the component-wise composition and identities, form the category* $\mathbf{AttC}^{\mathbf{F},\mathcal{A}}_{\mathbf{Atts}}$*. If the setting is clear, we may also write* $\mathbf{AttC}$*.*

Theorem 1. *The category* $\mathbf{AttC}$ *is well-defined, i.e. it is actually a category.*

Graphical Notation
Since the user does not want to know all the definitions and formal notations, attributed objects and morphisms are depicted in an UML-like notion, where the attributes are written down below the element they belong to as we already used in Fig. 4. Undefined attributes are denoted by □.

4 Transformations of Attributed Models

To apply transformations in attributed objects, we need to define rules and in particular rule morphisms. For our rules, undefined attributes shall only occur in the gluing part, but not in the left or right hand side. Such undefined attributes have to be concretized. Moreover, the data should not be changed by a rule, i.e.

the attribution value morphisms should be isomorphic. The morphism class $\mathcal{R}$ represents these available morphisms.

Definition 8 (Morphism class $\mathcal{R}$). *Given an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M}_\mathbf{C})$, we define the morphism class $\mathcal{R}$ in* **AttC** *by $\mathcal{R} = \{f : AO_1 \to AO_2 \mid f = (g, a) \in \mathbf{AttC}, g \in \mathcal{M}_\mathbf{C}, a$ is isomorphism, $AO_2 = (C_2, A_2, att_2), att_2$ is total$\}$.*

Unfortunately, the category **AttC** together with the morphism class $\mathcal{R}$ does not become an $\mathcal{M}$-adhesive category, because pushouts over $\mathcal{R}$-morphisms are not constructed preserving the morphism class $\mathcal{R}$.

Fact 1. *The category $(\mathbf{AttC}, \mathcal{R})$ is not an $\mathcal{M}$-adhesive category.*

Proof. The diagram in Fig. 6 shows that $\mathcal{M}$-morphisms are not stable under pushouts in **AttC**. We look at the graph with one node `n` of type `Cats`, where f_1 and f_2 are identities with total codomain attribution and thus in $\mathcal{R}$, and only the actual attribution of `n` is concretized. To obtain a pushout, we need a commuting square, i.e. its graph has to contain at least one node, where the nodes from B and C map to, which also has to be of type `Cats`. For the attribution values, `"Greebo"` and `"Maurice"` have to be merged by a_3 and a_4 – otherwise, either f_3 or f_4 will not be a valid attributed morphism. Actually, the depicted attributed graph is the pushout of f_1 and f_2, but $f_3, f_4 \notin \mathcal{R}$. Therefore, **AttC** with the chosen $\mathcal{R}$ is no $\mathcal{M}$-adhesive category.

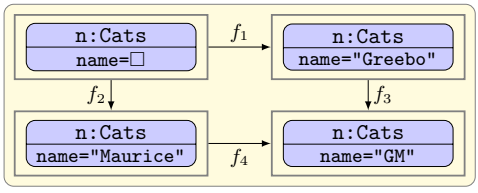


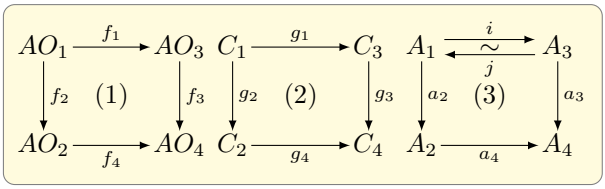
Fig. 6. Pushout in attributed graphs

Since our morphism class $\mathcal{R}$ is rather restrictive, one may argue that a more general morphism class may be suitable for **AttC** to become an $\mathcal{M}$-adhesive category. Thus we define the morphism class $\mathcal{M}$ as morphisms that are $\mathcal{M}_\mathbf{C}$-morphisms in the **C**-component and isomorphisms for the attribution values.

Definition 9 (Morphism class $\mathcal{M}$). *Given an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M}_\mathbf{C})$, we define the morphism class $\mathcal{M}$ in* **AttC** *by $\mathcal{M} = \{f : AO_1 \to AO_2 \mid f = (g, a) \in \mathbf{AttC}, g \in \mathcal{M}_\mathbf{C}, a$ is isomorphism$\}$.*

But even $(\mathbf{AttC}, \mathcal{M})$ is not an $\mathcal{M}$-adhesive category. Fig. 6 represents a counterexample, because $\mathcal{R} \subseteq \mathcal{M}$, but $f_3, f_4 \notin \mathcal{M}$. Nevertheless, the morphism class $\mathcal{M}$ will be useful for our further analysis. The above example gives us an idea how pushouts in **AttC** along $\mathcal{M}$-morphisms are constructed. Basically, we use the pushout construction in **C**, while we have to integrate different data values which are reached from the same elements by the given morphisms.

Fact 2 (Pushouts along $\mathcal{M}$). *The category* **AttC** *has pushouts along $\mathcal{M}$-morphisms.*

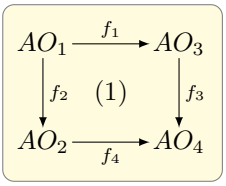


Construction. Consider the attributed morphisms $f_1 : AO_1 \to AO_3 \in \mathcal{M}$ and $f_2 : AO_1 \to AO_2$ with the single components as shown in the right diagrams, where due to the definition of $\mathcal{M}$ i and j are corresponding isomorphisms. We construct the pushout (2) in $\mathbf{C}$ and define a relation $\sim = \{(att_{2,k}(F_k(g_2)(o), n), a_2(j(att_{3,k}(F_k(g_1)(o), n)))) \in A_2 \times A_2 \mid k \in \mathcal{K}, n \in att_{name}(k), \exists o \in F_k(C_1), (o, n) \notin Dom(att_{1,k}), (F_k(g_2)(o), n) \in Dom(att_{2,k}), (F_k(g_1)(o), n) \in Dom(att_{3,k})\}$. Let $\equiv$ be the equivalence closure of $\sim$ and define $A_4 = A_2|_{\equiv}$. Now let $a_3(x) = [a_2(j(x))]$ and $a_4(x) = [x]$. Then $AO = (C_4, A_4, att_4)$ with $att_{4,k}(o, n) =$

$$\begin{cases} [a_2(j(att_{3,k}(o_3, n)))] & \exists o_3 \in F_k(C_3) : F_k(g_3)(o_3) = o, (o_3, n) \in Dom(att_{3,k}) \\ [att_{2,k}(o_2, n)] & \exists o_2 \in F_k(C_2) : F_k(g_4)(o_2) = o, (o_2, n) \in Dom(att_{2,k}) \\ \text{undefined} & \text{otherwise} \end{cases}$$

is the pushout object with morphisms $f_3 = (g_3, a_3)$ and $f_4 = (g_4, a_4)$.

Note that $\equiv$ is empty if each undefined attribute in AO_1 is also undefined in AO_2 or AO_3. In this case, A_4 is the pushout object of i and a_2 in **Sets**.
While the pushout construction is well-defined if one of the given morphisms is in $\mathcal{M}$, we do not want to glue attributes as done in Fig. 6 when applying a rule. Intuitively, gluing in the pushout construction has to be done for the following case: $\begin{smallmatrix} \square & \to & d \\ \downarrow & & \\ d' & & \end{smallmatrix}$. For transformations and their well-definedness, the key observation is the fact that such a situation could never occur. It is prevented because we have to construct a certain pushout complement first when applying a rule. Only the situations $\begin{smallmatrix} d & \leftarrow & \square & \to & d' \\ \downarrow & & \downarrow & & \downarrow \\ a(d) & \leftarrow & \square & \to & a(d') \end{smallmatrix}$ and $\begin{smallmatrix} d & \leftarrow & d & \to & d \\ \downarrow & & \downarrow & & \downarrow \\ a(d) & \leftarrow & a(d) & \to & a(d) \end{smallmatrix}$ can occur in the rule span and the corresponding application via a match. The construction of the pushout complement ensures that an undefined attribute in the intermediate object of the rule leads to an undefined attribute in the intermediate object of the rule application such that no data gluing has to occur. We describe such a well-behaviour as a special property and show that we can define a well-behaved pushout complement such that the double pushout of a direct transformation is actually well-behaved.



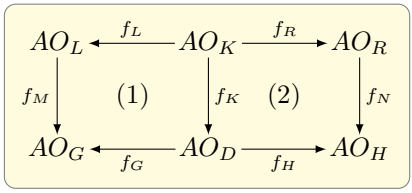
Definition 10 (Well-behaved pushouts). *The class $\mathcal{W}$ of morphism pairs with the same domain is defined by:* $\mathcal{W} = \{(f_1, f_2) \mid f_1 : AO_1 \to AO_3 \in \mathcal{R}, f_2 : AO_1 \to AO_2, \forall (o_1, n) \notin Dom(att_{1,k}) : (F_k(g_2)(o_1), n) \notin Dom(att_{2,k}), \forall o \in F_k(C_2) \backslash F_k(g_2)(C_1) : (o, n) \in Dom(att_{2,k})\}$.

A pushout (1) *is called* well-behaved*, or short a $\mathcal{W}$-pushout, if* $(f_1, f_2) \in \mathcal{W}$.

Note that the pushout over $(f_1, f_2) \in \mathcal{W}$ always exists, since **AttC** has pushouts along $\mathcal{M}$-morphisms (Fact 2). For $\mathcal{W}$-pushouts, in the construction of Fact 2 the relation $\sim$ is empty leading to $A_2 = A_4$, or $A_2 \cong A_4$ in general.

Now we can define rules and rule applications, which are called direct transformations, based on $\mathcal{W}$-pushouts. In contrast to rules for typed attributed graphs in [5], all attributes of the elements in the left or right hand side have to be defined, i.e. rules cannot be underspecified. While this requires some additional specification effort to at least assign a variable to each of these attributes, it clarifies the rule and is necessary for our approach. As a consequence, rules cannot be applied to attributed objects with only partial attribution.

Definition 11 (Rule and direct transformation). *A rule is a span* $p = (AO_L \xleftarrow{f_L} AO_K \xrightarrow{f_R} AO_R)$ *of* $\mathcal{R}$*-morphisms in* **AttC**.



Given a rule p*, an attributed object* AO_G*, and an attributed morphism* $f_M : AO_L \to AO_G$*, called match, a direct transformation* $AG \xRightarrow{p,f_M} AH$ *is given by two* $\mathcal{W}$*-pushouts* (1) *and* (2) *as in the diagram on the right.*

Note that we could pragmatically adapt the rules of typed attributed graphs in [5] with conditions as expressed by $\mathcal{R}$-morphisms. These conditions can be checked statically to allow only intended attribute changes. Nevertheless, the underlying graph model still inhibits the disadvantages explained in Section 2.

Similar to the standard graph transformation, we define the gluing condition which characterizes valid situations, where the pushout complement exists.

Definition 12 (Gluing condition). *Given* f_1 *and* f_3 *as in Def. 10, the* gluing condition *holds if the underlying pushout complement of* $g_3 \circ g_1$ *in* **C** *exists and for all* $o_1, o_1' \in F_k(C_1)$ *with* $F_k(g_3 \circ g_1)(o_1) = F_k(g_3 \circ g_1)(o_1')$ *we have that both* $att_{1,k}(o_1, n).att_{1,k}(o_1', n)$ *are either defined or undefined.*

Since both $\begin{smallmatrix} d & \leftarrow & \square \\ \downarrow & & \downarrow \\ d & \leftarrow & d \end{smallmatrix}$ and $\begin{smallmatrix} d & \leftarrow & \square \\ \downarrow & & \downarrow \\ d & \leftarrow & \square \end{smallmatrix}$ are valid pushout complements, pushout complements in **AttC** are not unique, but only the second situation behaves well. Thus, we chose this one to construct pushout complements in **AttC**.

Fact 3 (Pushout complement along $\mathcal{M}$). *The category* **AttC** *has pushout complements along* $\mathcal{M}$*-morphisms, if the gluing condition holds.*

Construction. Given attributed morphisms $f_1 : AO_1 \to AO_3 \in \mathcal{M}$ and $f_3 : AO_3 \to AO_4$ as above, construct the pushout complement (2) in **C** (see Construction of Fact 2). Now define $A_2 = A_4$, $a_2 = a_3 \circ i$, $a_4 = id$, and $att_{2,k}(o, n) =$

$$\begin{cases} a_2(att_{1,k}(o_1, n)) & \exists o_1 \in F_k(C_1) : F_k(g_2)(o_1) = o, (o_1, n) \in Dom(att_{1,k}) \\ \text{undefined} & \exists o_1 \in F_k(C_1) : F_k(g_2)(o_1) = o, (o_1, n) \notin Dom(att_{1,k}) \\ att_{4,k}(F_k(g_4)(o), n) & \text{otherwise} \end{cases}.$$

Then $AO_2 = (C_2, A_2, att_2)$ together with the morphisms $f_2 = (g_2, a_2)$ and $f_4 = (g_4, a_4)$ is the pushout complement of f_1 and f_3.

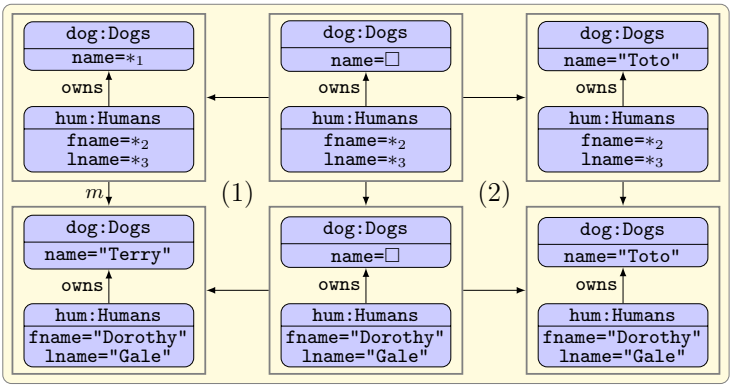


Fig. 7. Direct transformation of attributed graphs

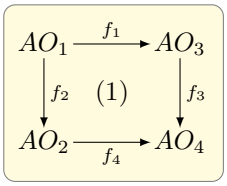
A rule p is applicable to AO_G if the gluing condition holds and the attribution in AO_G is total. In this case, for the construction of a direct transformation we can first construct the pushout complement of Fact 3 and then the pushout of Fact 2, which are both unique and well-behaved.

Theorem 2 (Construction of direct transformation). *Given a situation as in Def. 11 with $f_L, f_R \in \mathcal{R}$, where the gluing condition holds and att_G is total, then the pushout complement* (1) *and the pushout* (2) *are uniquely defined $\mathcal{W}$-pushouts. Moreover, also $f_G, f_H \in \mathcal{R}$.*

Example 4. Now we can rename `"Greebo"` to `"Maurice"` using the second rule in Fig. 2 or an arbitrary dog that belongs to someone to `"Toto"` with the rule in the top row of Fig. 7. When applying such a rule, we construct first the pushout complement and then the pushout, as shown in Fig. 7. Note that the match m uses the variable assignment $\alpha(*_1) =$ `"Terry"`, $\alpha(*_2) =$ `"Dorothy"`, and $\alpha(*_3) =$ `"Gale"`, and its codomain graph has total attribution. We can see that both pushouts are well-behaved because all attributes in the intermediate object that are undefined in AO_K are also undefined in AO_D.

As we will need this fact in the next section, we state that also pullbacks along $\mathcal{M}$-morphisms exist in **AttC**.

Fact 4 (Pullbacks over $\mathcal{M}$). *The category* **AttC** *has pullbacks along $\mathcal{M}$-morphisms. Moreover, $\mathcal{M}$ is closed under pullbacks.*



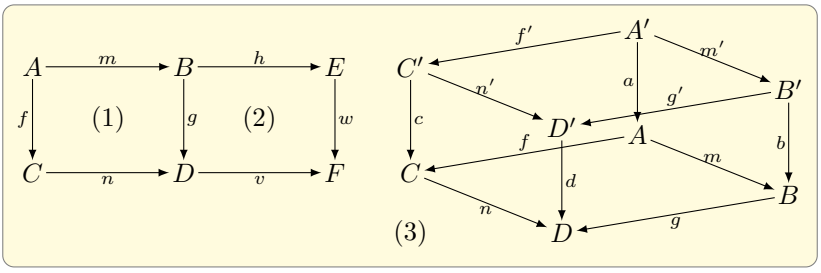
Construction. Given $f_4 \in \mathcal{M}$ we construct the pullback in **C** with pullback object C_1 and morphisms g_1, g_2. Then $AO_1 = (C_1, A_3, att_1)$ with $f_1 = (g_1, id_{A_3})$, $f_2 = (g_2, a_4^{-1} \circ a_3)$, and $att_1(o, n) =$ $\begin{cases} att_{3,k}(F_k(g_1)(o), n) & (F_k(g_1)(o), n) \in Dom(att_{3,k}), \\ & (F_k(g_2)(o), n) \in Dom(att_{2,k}) \\ \text{undefined} & \text{otherwise} \end{cases}$ is the pullback of f_3 and f_4.

5 $\mathcal{W}$-Adhesive Categories

As we have analyzed in the last section, not all pushouts are important in the context of transformations, but only those we have called $\mathcal{W}$-pushouts. In this section, we generalize this idea to $\mathcal{W}$-adhesive categories, which are categories where the VK property is restricted to hold for $\mathcal{W}$-pushouts, which are defined by a class $\mathcal{W}$ of morphism spans. This restriction allows us to formulate a transformation theory for the category **AttC**.

Definition 13 ($\mathcal{W}$-adhesive category). *Given a category* $\mathbf{C}$*, morphism classes* $\mathcal{R} \subseteq \mathcal{M}$*, and a class* $\mathcal{W} \subseteq \mathcal{R} \times Mor_{\mathbf{C}}$ *of morphism spans,* $(\mathbf{C}, \mathcal{R}, \mathcal{M}, \mathcal{W})$ *is a* $\mathcal{W}$-adhesive category *if:*

1. $\mathcal{M}$ *is a class of monomorphisms closed under isomorphisms, composition, and decomposition, with* $id_A \in \mathcal{M}$ *for all* $A \in \mathbf{C}$*,*
2. $\mathbf{C}$ *has pushouts along and pullbacks over* $\mathcal{M}$*-morphisms,*
3. $\mathcal{M}$ *is closed under pullbacks: Given a pullback* (1) *with* $n \in \mathcal{M}$ *then also* $m \in \mathcal{M}$*,*
4. $\mathbf{C}$ *has pushouts over* $\mathcal{W}$*-morphisms, called* $\mathcal{W}$*-pushouts,*
5. $\mathcal{R}$ *is closed under* $\mathcal{W}$*-pushouts: Given* $\mathcal{W}$*-pushout* (1) *with* $m \in \mathcal{R}$ *then also* $n \in \mathcal{R}$*,*
6. $\mathcal{W}$ *is closed under* $\mathcal{R}$*:* $(m' : A' \to B', a) \in \mathcal{W}$*,* $f' : A' \to C' \in \mathcal{R}$ *implies* $(f', a) \in \mathcal{W}$*,*
7. $\mathcal{W}$*-pushout composition and decomposition: Given pushout* (1)*,* (1) + (2) *is a* $\mathcal{W}$*-pushout with* $(f, h \circ m) \in \mathcal{W}$ *if and only if* (2) *is a* $\mathcal{W}$*-pushout with* $(g, h) \in \mathcal{W}$*,*
8. $\mathcal{W}$*-pushouts fulfill the* $\mathcal{W}$*-van Kampen property: Given a commutative cube* (3) *with* $\mathcal{W}$*-pushout* (1) *in the bottom,* $m, d \in \mathcal{R}$*,* $b, c \in \mathcal{M}$ *and the back faces being pullbacks, it holds that the top is a pushout if and only if the front faces are pullbacks.*



An example for a $\mathcal{W}$-adhesive category is the category of attributed objects and morphisms.

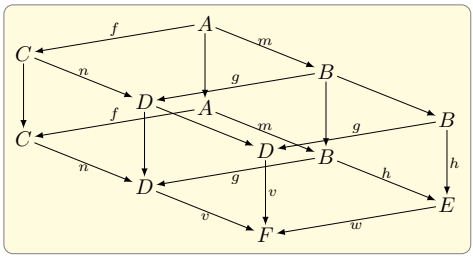
Theorem 3. AttC *with an underlying* $\mathcal{M}$*-adhesive category* $(\mathbf{C}.\mathcal{M}_{\mathbf{C}})$*,* $\mathcal{R}$ *as defined in Def. 8,* $\mathcal{M}$ *as defined in Def. 9, and* $\mathcal{W}$ *as defined in Def. 10 is a* $\mathcal{W}$*-adhesive category.*

Proof. Obviously, $\mathcal{M}$ is a class of monomorphisms because both components are monomorphisms with the required closure properties inherited from the components. **AttC** has pullbacks over $\mathcal{M}$-morphisms and $\mathcal{M}$ is closed under pullbacks as shown in Fact 4. Since **AttC** has pushouts along $\mathcal{M}$-morphisms, as shown in Fact 2, it has pushouts over $\mathcal{M}$-morphisms as well as $\mathcal{W}$-pushouts. The closure of $\mathcal{R}$ under $\mathcal{W}$-pushouts and the closure of $\mathcal{W}$ under $\mathcal{R}$ follow from Thm. 2. The $\mathcal{W}$-pushout composition and decomposition follows from the fact that the undefined attributes in A, B, and E are exactly the same. The proof of the $\mathcal{W}$-van Kampen property can be found in the appendix.

As specified in Def. 11 for the special case of **AttC**, in a $\mathcal{W}$-adhesive category rules are defined as spans of $\mathcal{R}$-morphisms, while transformations are defined by double $\mathcal{W}$-pushouts.

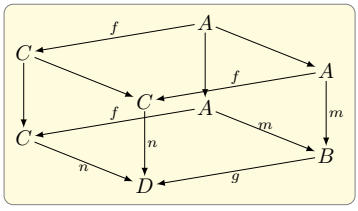
To prove important results for graph transformation, various so-called HLR properties have been used in [5]. Here, we show the corresponding variant for $\mathcal{W}$-categories for two of them, the $\mathcal{W}$-pushout-pullback decomposition property and that $\mathcal{W}$-pushouts are pullbacks.

Fact 5 ($\mathcal{W}$-pushout-pullback decomposition). *Given the above commutative diagram, where* (1) + (2) *is a $\mathcal{W}$-pushout,* (2) *is a pullback, $v \in \mathcal{R}$, and* ($f \in \mathcal{R}$ *or* $m, h \circ m \in \mathcal{R}$)*, then* (1) *is a pushout.*



Proof. Consider the right cube, where all unnamed morphisms are identities. The bottom is the $\mathcal{W}$-pushout (1) + (2) with ($f \in \mathcal{R}$ or $m, h \circ m \in \mathcal{R}$), $v \in \mathcal{R}$, and $h, id_B, id_C, id_D \in \mathcal{M}$ (Def. 13 Items 3 and 1). All back and front faces are pullbacks. Now the $\mathcal{W}$-van Kampen property (Item 8) implies that the top, i.e. the square (1), is a pushout.

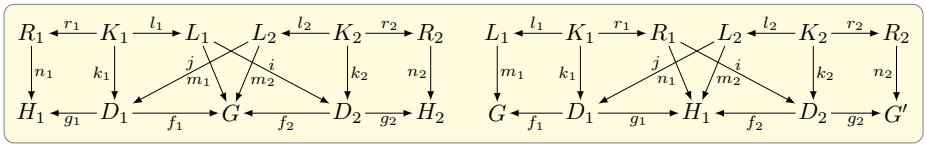
Fact 6 ($\mathcal{W}$-pushouts are pullbacks). *Given the above $\mathcal{W}$-pushout* (1) *with $m \in \mathcal{R}$, then* (1) *is also a pullback.*



Proof. Consider the right cube, where all unnamed morphisms are identities. The bottom is the $\mathcal{W}$-pushout (1) with $m, n \in \mathcal{R}$ (Def. 13 Item 5), and $m, id_C \in \mathcal{M}$ (Item 1). All back faces are pullbacks and the top is a pushout. Now the $\mathcal{W}$-van Kampen property (Item 8) implies that the front faces, and in particular the square (1), are pullbacks.

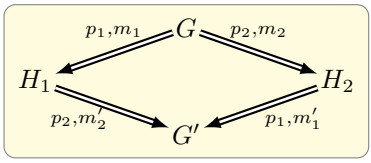
In the following, we sketch for the example of (one direction of) the Local Church-Rosser Theorem that this result is also available in $\mathcal{W}$-adhesive categories. It is concerned with parallel and sequential independence of direct transformations. First, we define the notion of parallel and sequential independence. Then we state the Local Church-Rosser Theorem and prove it. The proof follows the one in [5] with certain adaptions for $\mathcal{W}$-adhesive categories.

Definition 14 (Parallel and sequential independence). *Two direct transformations* $G \overset{p_1,m_1}{\Longrightarrow} H_1$ *and* $G \overset{p_2,m_2}{\Longrightarrow} H_2$ *are* parallel independent *if there are morphisms* $i : L_1 \to D_2$ *and* $j : L_2 \to D_1$ *such that* $f_2 \circ i = m_1$ *and* $f_1 \circ j = m_2$.

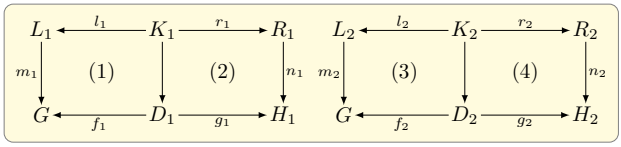


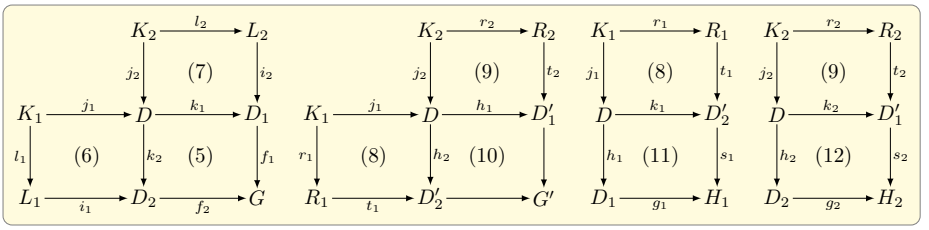
Two direct transformations $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2}{\Longrightarrow} G'$ *are* sequentially independent *if there are morphisms* $i : R_1 \to D_2$ *and* $j : L_2 \to D_1$ *such that* $f_2 \circ i = n_1$ *and* $g_1 \circ j = m_2$.

Theorem 4 (Local Church-Rosser Theorem). *Given two parallel independent direct transformations* $G \overset{p_1,m_1}{\Longrightarrow} H_1$ *and* $G \overset{p_2,m_2}{\Longrightarrow} H_2$ *there is an object* G' *together with direct transformations* $H_1 \overset{p_2,m_2'}{\Longrightarrow} G'$ *and* $H_2 \overset{p_1,m_1'}{\Longrightarrow} G'$ *such that* $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} G'$ *and* $G \overset{p_2,m_2}{\Longrightarrow} H_2 \overset{p_1,m_1'}{\Longrightarrow} G'$ *are sequentially independent.*

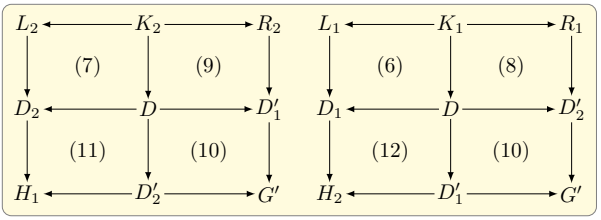


Proof. Consider the parallel independent direct transformations $G \overset{p_1,m_1}{\Longrightarrow} H_1$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2$ depicted right. We combine the $\mathcal{W}$-pushouts (1) and (3) with the morphisms i_1 and i_2 obtained by parallel independence. Since $f_1, f_2 \in \mathcal{M}$ (Def. 13 Item 5) we can construct the pullback (5) (Item 2) and obtain morphisms j_1 and j_2 as shown in the following diagram on the left-hand side. Since (1) = (6) + (5) with $f_2, l_1 \in \mathcal{R}$ Fact 5 implies that (6), and analogously (7), is a pushout. Now we construct the pushouts (8) and (9) (Item 2) along $r_1, r_2 \in \mathcal{M}$. Finally, the pushout (10) is constructed by decomposition of pushout (8) and the pushout over $r_1 \in \mathcal{M}$ and $h_1 \circ j_1$ (Item 2). From pushout (8) we obtain a morphism $s_1 : D_2' \to H_1$ such that (2) = (8) + (11), and by Item 7 (11) is a $\mathcal{W}$-pushout. Analogously, we obtain the $\mathcal{W}$-pushout (12).





By Item 7, (7) + (11) and (6) + (12) are $\mathcal{W}$-pushouts, and using Item 6 we obtain the sequentially independent direct transformations $H_1 \overset{p_2}{\Longrightarrow} G'$ and $H_2 \overset{p_1}{\Longrightarrow} G'$.



6 Conclusion and Future Work

In this paper, we proposed a new concept for attribution of objects in an arbitrary category based on a functor selecting attributable elements of objects and assigning attributes and values to them. We have then defined rules and transformations on attributed objects based on $\mathcal{M}$-adhesive categories, where transformations rely on the new concept of $\mathcal{W}$-pushouts. This concept leads to the notion of $\mathcal{W}$-categories, which are a suitable framework to show results for transformations of attributed objects, as we have demonstrated for the example of the Local Church–Rosser Theorem. Since the underlying objects come from an $\mathcal{M}$-adhesive category, we only have to prove the results for the attribution part and can rely on the underlying results for the pure, un-attributed transformations. In fact, we can use any suitable category for attribution. In this paper, we chose $\mathcal{M}$-adhesive categories, because a large number of results is available there, but there is actually no need for this restriction.

$\mathcal{W}$-adhesive categories have been introduced for attribution in this paper, but they may also fit for transformations in other non-$\mathcal{M}$-adhesive categories, where the definition of the proper pushout complement depends on both morphisms, like in RDF graphs (see [13]) or open Petri nets (see [14]). In this sense, they are more expressive then $\mathcal{M}$-$\mathcal{N}$-adhesive categories [15], which consider a special class $\mathcal{N}$ for vertical morphisms in transformations.

Future work includes to prove other important results and theorems for graph transformation, where additional requirements for $\mathcal{W}$-adhesive categories may have to be identified. For example, for the Parallelism Theorem some compatibility property of $\mathcal{W}$ with binary coproducts will be necessary. Moreover, rules should be extended with constraints and application conditions for data as done in [16] for triple rules. Another interesting extension would be to integrate an inheritance concept for the attribution types.

For language evolution, the deletion or addition of attributes, i.e. changing the meta-model, is an interesting field of work. Without the restrictions of the

$\mathcal{R}$-morphisms in the rules, especially the total attribution, we would not be able to define unique pushout complements for a transformation. It would be interesting to explore if other graph transformation approaches may require fewer restrictions to the rule morphisms. For computations and reasoning on attributes, symbolic graphs [8] can be adapted for the underlying $\mathcal{M}$-adhesive category, and their usefulness should be further elaborated.

References

1. Ehrig, H.: Introduction to the Algebraic Theory of Graph Grammars (A Survey). In: Ng, E.W., Ehrig, H., Rozenberg, G. (eds.) Graph Grammars 1978. LNCS, vol. 73, pp. 1–69. Springer, Heidelberg (1979)
2. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformation. Foundations, vol. 1. World Scientific (1997)
3. Ehrig, H., Golas, U., Hermann, F.: Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach. BEATCS 102, 111–121 (2010)
4. Lack, S., Sobociński, P.: Adhesive Categories. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 273–288. Springer, Heidelberg (2004)
5. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs. Springer (2006)
6. Hermann, F., Ehrig, H., Ermel, C., Orejas, F.: Concurrent Model Synchronization with Conflict Resolution Based on Triple Graph Grammars. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 178–193. Springer, Heidelberg (2012)
7. Golas, U.: A General Attribution Concept for Models in M-adhesive Transformation Systems: Long Version. Technical Report 12-22, Zuse Institute Berlin (2012)
8. Orejas, F., Lambers, L.: Symbolic Attributed Graphs for Attributed Graph Transformation. ECEASST 30 (2010)
9. Habel, A., Plump, D.: Relabelling in Graph Transformation. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 135–147. Springer, Heidelberg (2002)
10. Löwe, M., Korff, M., Wagner, A.: An Algebraic Framework for the Transformation of Attributed Graphs. In: Term Graph Rewriting: Theory and Practice, pp. 185–199. Wiley (1993)
11. Rebout, M., Féraud, L., Soloviev, S.: A Unified Categorical Approach for Attributed Graph Rewriting. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) CSR 2008. LNCS, vol. 5010, pp. 398–409. Springer, Heidelberg (2008)
12. Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification 1: Equations and Initial Semantics. EATCS Monographs. Springer (1985)
13. Braatz, B., Brandt, C.: Graph Transformations for the Resource Description Framework. ECEASST 10 (2008)
14. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional Modeling of Reactive Systems Using Open Nets. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 502–518. Springer, Heidelberg (2001)
15. Habel, A., Plump, D.: *M,N*-Adhesive Transformation Systems. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2012. LNCS, vol. 7562, pp. 218–233. Springer, Heidelberg (2012)
16. Anjorin, A., Varro, G., Schürr, A.: Complex Attribute Manipulation in TGGs with Constraint-Based Programming Techniques. ECEASST (2012)