# Graph Attribution Through Sub-Graphs

Harmen Kastenberg and Arend Rensink(✉)

Department of Computer Science, University of Twente, Enschede, The Netherlands
arend.rensink@utwente.nl

**Abstract.** We offer an alternative to the standard way of formalising attributed graphs. We propose to represent them as graphs with a marked sub-graph that represents the data domain, rather than as tuples of graph and algebra. This is a general construction which can be shown to preserve adhesiveness of categories; it has the advantage of uniformity and gives more flexibility in defining data abstractions. We show equivalence of our formalisation with the standard one, under a suitable encoding of algebras as graphs.

## 1 Introduction

Graph transformation has many strengths and pleasant characteristics, but the treatment of data values, such as integers, booleans and strings, is not among them. In fact, the core idea of graph-based modelling is that concrete node and edge identities are irrelevant, and so graphs can be regarded up to isomorphism; this, however, is simply no longer true if the nodes stand for data values.

Nevertheless, the large majority of systems for which graph-based modelling is appropriate do include primitive data, in the form of attributes. There is therefore no question but that graph transformation has to cope with data in order to be practically useful in modelling real-world applications. And so, a model for attributed graphs has been worked out by Ehrig et al. [3], which we will refer to as the *standard model*.

The standard model explicitly combines the world of graphs and that of algebras; the manipulation of the data is deferred to the second, whereas the data values appear as nodes in the graphs, to which it is possible to define edges from ordinary nodes. Such edges then stand for attributes. Although this is theoretically satisfactory, in that the model allows us to use attributes, and is, moreover, a "nice" category for graph transformation (meaning that it is adhesive HLR — more about this later), we feel that the standard model leaves some things to be desired.

– Due to the presence of both graphs and algebras in the standard model, some things are solved twice. In particular, in transformation rules, the algebra component uses variables, terms and (in)equations, whereas the graph component uses nodes, edges and (non)injectivity constraints, for essentially the same functionality. This means that users have two different formalisms to

© Springer International Publishing AG, part of Springer Nature 2018
R. Heckel and G. Taentzer (Eds.): Ehrig Festschrift, LNCS 10800, pp. 245–265, 2018.
https://doi.org/10.1007/978-3-319-75396-6_14

cope with, and the visual presentation of rules needs to combine graphical and textual parts. Moreover, an implementation also needs to contain distinct algorithms for matching the graph and algebra parts.
– We are studying abstraction in graph transformation, in particular also data abstraction. A very limited form of abstraction is possible using algebras, by moving from the standard algebra of a given signature (for instance, the integers with successor, addition and multiplication) by a surjective homomorphism to another algebra (for instance, the integers modulo an upper bound). However, many interesting abstractions cannot be formulated as algebra homomorphisms. For instance, the classical abstraction of the integers into the three-valued set of "strict negative", "zero" and "strict positive" either does not give rise to an algebra (the operations are not deterministic); or, if we add the joined elements "negative", "positive" and "all", then there is no homomorphism from the standard algebra to this one.

The first of these issues has prompted us to consider a version of attributed graphs in which the algebras are entirely encoded as sub-graphs. In particular, the operations are also coded up, by adding corresponding nodes and edges. A preliminary version of this idea was presented in [7]. Since (at need) these sub-graphs are easily distinguishable from the surrounding "real" graphs by typing, in most circumstances we can proceed as if we were dealing with standard graphs.

A side benefit is that this sub-graph arrangement can be understood as a general categorical construction: namely, it gives rise to a category of *reflected monos*, in which the objects are monos (corresponding to embedded graphs) and the arrows are pullbacks. The proof of adhesiveness of the resulting category can therefore be established on a more general level than for the standard model.

It turns out that this also provides a solution to the second issue. By extending the set of "algebra graphs" allowed as sub-graphs with graphs in which the algebraic operations are not deterministic (and so are no longer truly operations), we can easily cope with data abstractions such as the one mentioned above. Our proof of adhesiveness carries over to the extended category without any changes. Now the embedding theorem implies that the abstract graphs over-approximate the behaviour of the concrete graphs. We also extend the embedding theorem to rules with negative application conditions, provided that these do not test (negatively) for the data part.

The paper is structured as follows: in Sect. 2 we define our attributed graph category and establish equivalence with the standard model. In Sect. 3 we give an independent proof that the construction gives rise to an adhesive category. In Sect. 4 we discuss data abstraction, and we show that the embedding theorem still holds in the presence of NACs which do not test for data. In Sect. 5 we briefly discuss the implementation of these concepts in the graph transformation tool GROOVE. Section 6 concludes the paper.

Almost all of the proofs are silently omitted from this version of the paper. For the full technical report, including all proofs, see [8].

## 2 The Model

In this section, we show how the structure of any algebra can be encoded as a graph. We then combine these *algebra graphs* with the graphs that need attribution, giving rise to larger graphs of which the algebra graphs are sub-graphs; attributes then take the form of edges from the surrounding graph into the algebra sub-graph.

Some general notational conventions: if $s \in A^*$ is a sequence, say $s = s_1 \cdots s_n$, then $|s|$ denotes the length ($n$), $[s]$ denotes the set of elements in $s$ ($\{s_1, \ldots, s_n\}$), and for all $1 \leq i \leq n$, $s|_i$ denotes the $i$th element ($s_i$). The empty sequence is denoted $\varepsilon$.

### 2.1 Algebra Graphs

Let us first recall the standard definitions of signatures and algebras. We assume a global set **Name** of names, which are symbols that are of themselves uninterpreted; the interpretation is given by their use.

**Definition 1 (signature).** *A* signature *is a tuple* $\Sigma = \langle S, O, \sigma, \tau \rangle$ *where* $S \subseteq$ **Name** *is a set of* sorts*,* $O \subseteq$ **Name** *is a set of* operators*, disjoint from* $S$*,* $\sigma \colon O \to S^*$ *is the* source typing *of the operators, and* $\tau \colon O \to S$ *is the* target typing *of the operators.*

We call a sort $s$ of a given signature *spurious* if there is no operator that uses it, i.e., $s \notin [\sigma(o)]$ for all $o \in O$. In this paper we assume that signatures have no spurious sorts.

Given a signature, the *arity* of an operator $o \in O$ is given by $\alpha(o) = |\sigma(o)|$. We call a signature *unary* if $\alpha(o) = 1$ for all $o \in O$.

*Example 2.* As a running example we use the algebra of booleans and integers with a few operations. This is given by the signature Prim with $S = \{\mathsf{Int}, \mathsf{Bool}\}$ and $O$, $\sigma$ and $\tau$ given by the following table. (lt stands for *lesser than.*)

| $O$ | zero | succ | pred | add | lt | pos | true | false | not |
|---|---|---|---|---|---|---|---|---|---|
| $\sigma$ | $\varepsilon$ | Int | Int | Int Int | Int Int | Int | $\varepsilon$ | $\varepsilon$ | Bool |
| $\tau$ | Int | Int | Int | Int | Bool | Bool | Bool | Bool | Bool |

**Definition 3 (algebra).** *An* algebra *over a signature* $\Sigma$ *is a tuple* $A = \langle D, F \rangle$ *where*

- $D = (D^s)_{s \in S}$ *is an* $S$*-indexed family of disjoint* data sets*;*
- $F = (f^o)_{o \in O}$ *is an* $O$*-indexed family of* functions *typed by the signature; i.e., for all* $o \in O$*, if* $\sigma(o) = s_1 \cdots s_n$ *then* $f^o \colon D^{s_1} \times \cdots \times D^{s_n} \to D^{\tau(o)}$*.*

*Given two algebras* $A_i = \langle D_i, F_i \rangle$ *over* $\Sigma$ *(*$i = 1, 2$*), an* algebra morphism *is a family of functions* $h = (h^s \colon D_1^s \to D_2^s)_{s \in S}$ *such that for all* $o \in O$ *with* $\sigma(o) = s_1 \cdots s_n$ *and for all* $d_j \in D_1^{s_j}$ *(*$j = 1, \ldots, n$*):*

$$h^s(f_1^o(d_1, \ldots, d_n)) = f_2^o(h^{s_1}(d_1), \ldots, h^{s_n}(d_n)) \ .$$

We commonly use $D_A$ and $F_A$ to denote the data sets and functions of an algebra $A$; we omit the subscript $A$ if it is clear from the context. The algebras over a signature $\Sigma$ together with the algebra morphisms form a category, which we call $\mathbf{Alg}(\Sigma)$.

*Example 4.* For the signature of Example 2, one may consider the following algebras:

- The *initial* or *term* algebra $A_{\mathsf{Term}}$, where all terms built over $\mathsf{Prim}$ denote distinct elements. The data sets of this algebra consist of the (syntax trees of the) terms themselves.
- The *natural* or *standard* algebra $A_{\mathsf{Std}}$, consisting of the "real" integers and booleans.
- The *final* or *point* algebra $A_{\mathsf{Point}}$, where the data sets are all singletons, i.e., all values are collapsed to a single one.

There are unique algebra morphisms from $A_{\mathsf{Term}}$ to $A_{\mathsf{Std}}$ and from $A_{\mathsf{Std}}$ to $A_{\mathsf{Point}}$; for instance, if $h\colon A_{\mathsf{Term}} \to A_{\mathsf{Std}}$ then $h^{\mathsf{Int}}(\mathsf{succ}(\mathsf{zero}())) = 1$ and $h^{\mathsf{Bool}}(\mathsf{true}()) = h^{\mathsf{Bool}}(\mathsf{not}(\mathsf{false}())) = \mathit{true}$.

The encoding of algebras as graphs is essentially straightforward:

- The data values (i.e., the elements of the carrier sets) are represented by nodes
- The functions are interpreted as sets of pairs of elements from the function domain, respectively codomain; these pairs are then represented by edges.

The only complication is that, for operators with arity $> 1$, the domain of the corresponding function is a cartesian product; in order to interpret such a function as a set of edges we need to introduce nodes for the elements of the domain, i.e., nodes that stand for *tuples* of data values. For unary signatures, this complication does not arise, hence we concentrate on these first; we then show a way to transform algebras over arbitrary signatures into equivalent algebras over unary signatures.

**Definition 5 (graph).** *A* graph *is a tuple* $G = \langle N, E, src, tgt, lab\rangle$ *where* $N$ *is a set of* nodes*,* $E$ *is a set of* edges*,* $src\colon E \to N$ *is a* source function*,* $tgt\colon E \to N$ *is a* target function*, and* $lab\colon E \to \mathbf{Name}$ *is a* labelling*.*

*Given two graphs* $G_i = \langle N_i, E_i, src_i, tgt_i, lab_i\rangle$ *for* $i = 1, 2$*, a* graph morphism *from* $G_1$ *to* $G_2$ *is a pair* $h = (h^N\colon N_1 \to N_2, h^E\colon E_1 \to E_2)$ *such that, for all* $e \in E_1$*,*

$$\begin{aligned} src_2(h^E(e)) &= h^N(src_1(e)) \\ tgt_2(h^E(e)) &= h^N(tgt_1(e)) \\ lab_2(h^E(e)) &= lab_1(e). \end{aligned}$$

We commonly use $N_G, E_G$ etc. to denote the components of a graph $G$; we omit the subscript $G$ if it is clear from the context. Graphs and graph morphisms form a category, which we call **Graph** (identity arrows are pairs of identity functions

over the node and edge sets, and arrow composition is component-wise composition of the node and edge functions). We call a graph $G$ *discrete* if $E_G = \emptyset$, i.e., the graph consists of nodes only. The full sub-category of **Graph** consisting of discrete graphs will be denoted **dGraph**. Note that a unary signature $\Sigma$ can be seen as a graph where the nodes are sorts and the edges are operators. For edge labels we can use the operators themselves. This gives rise to the *signature graph* $G_\Sigma = \langle S, O, \sigma, \tau, id_O \rangle$.

**Definition 6 (algebra graph).** *Let $\Sigma$ be a unary signature. An* algebra graph *over $\Sigma$ is a graph $G$ with a morphism $t$ to $G_\Sigma$ such that for all $n \in N_G$ and $o \in O$, if $t^N(n) = \sigma(o)$ then there is an edge $e \in E_G$ such that $src(e) = n$ and $t^E(e) = o$. $G$ is called* deterministic *if this edge $e$ is always unique.*

For a given unary signature $\Sigma$, we use $\mathbf{AlgGraph}^+(\Sigma)$ to denote the full sub-category of **Graph** consisting of all algebra graphs over $\Sigma$, and $\mathbf{AlgGraph}(\Sigma)$ for the full (further) sub-category of deterministic algebra graphs.

(The upshot of the above definition is that $t$ acts as a typing morphism from $G$ to $G_\Sigma$; the additional conditions on the existence and, in the case of determinism, uniqueness of edges can be understood as multiplicity constraints in the type graph $G_\Sigma$: all edges have outgoing multiplicity $1..*$ or, in the case of determinism, 1.)

*Example 7.* Figure 1 shows an algebra graph for a variation on Prim, viz., the unary signature $\Sigma$ with $S_\Sigma = S_{\mathsf{Prim}}$ and $O_\Sigma = \{\mathsf{succ}, \mathsf{odd}, \mathsf{not}\}$. Here, odd tests if a number is odd; it has $\sigma(\mathsf{odd}) = \mathsf{Int}$ and $\tau(\mathsf{odd}) = \mathsf{Bool}$.

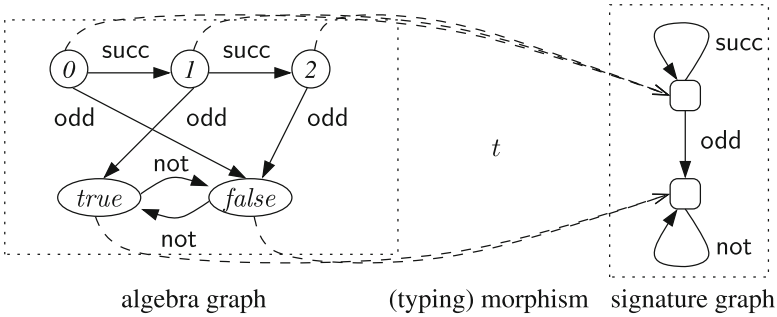



**Fig. 1.** Algebra graph with typing into the signature graph. Italic node labels stand for algebra values.

The following proposition is important in that it implies that it is enough to know that a graph is in $\mathbf{AlgGraph}^+(\Sigma)$ (for a given unary signature $\Sigma$) in order to reconstruct the actual typing morphism. This relies on our assumption that $\Sigma$ has no spurious sorts.

**Proposition 8.** *For any $\Sigma$ and $G \in \mathbf{AlgGraph}^+(\Sigma)$, there exists exactly one (typing) morphism $t\colon G \to G_\Sigma$.*

The following theorem essentially states that our encoding of algebras as graphs works.

**Theorem 9.** *For any unary $\Sigma$,* $\mathbf{Alg}(\Sigma)$ *and* $\mathbf{AlgGraph}(\Sigma)$ *are equivalent.*

This is proved by two functors, one of which turns data values into nodes and codes up the operations as edges, and the other of which undoes this by reconstructing the operations from the edges. The full proof can be found in [8].

For non-unary signatures, the situation is more complicated: first we have to *flatten* the signatures and algebras, but we also have to impose some additional constraints on the flattened algebras in order to get an equivalent category.

**Definition 10 (product sorts).**

1. *A* signature with products *is a pair $\Sigma|\pi$ where $\Sigma = \langle S, O\rangle$ is a unary signature and $\pi\colon S \rightharpoonup O^*$ is a partial function that assigns to some of the sorts (called the product sorts) a sequence of distinct* projection operators, *such that $src(o) = s$ for all $o \in [\pi(s)]$. For product sorts $p \in dom(\pi)$ we use $w(p) = |\pi(p)|$ to denote the* width *of $p$, and $\pi_{p,i}$ $(1 \leq i \leq w(p))$ to denote the individual elements of $\pi(p)$ (hence $\pi(p) = \pi_{p,1} \cdots \pi_{p,w(p)}$).*
2. *An* algebra over $\Sigma|\pi$ *is an algebra over $\Sigma$ such that, in addition, for all sorts $p \in dom(\pi)$ and all combinations of data values $(d_i \in D^{tgt(\pi_{p,i})})_{1\leq i\leq w(p)}$ from the target sorts of the projection operators, there is a unique $d \in D^p$ with $f^{\pi_{p,i}}(d) = d_i$ for all $1 \leq i \leq w(p)$.*
3. *An* algebra graph *$G$ over $\Sigma|\pi$ is an algebra graph over $\Sigma$, with typing $t$, such that, in addition, for all product sorts $p \in dom(\pi)$ and all combinations of nodes $(n_i \in t^{N,-1}(tgt(\pi_{p,i})))_{1\leq i\leq w(p)}$ typed by the target sorts of the projection operators, there is an $n \in N$ and a family of edges $(e_i \in E)_{1\leq i\leq w(p)}$ such that for all $1 \leq i \leq w(p)$, $t^E(e) = \pi_{p,i}$, $src(e) = n$ and $tgt(e) = n_i$. $G$ is called deterministic if, in addition to the conditions of Definition 6, this $n$ is unique.*

The underlying intuition is as follows: if $p$ is a product sort with projection operators $\pi(p) = o_1 \cdots o_n$, and respective target sorts $s_1 \cdots s_n$, then Clause 10.2 above guarantees that $D^p$ is essentially the cartesian product $D^{s_1} \times \cdots D^{s_n}$ and the $o_i$ project the values of $D^p$ to their $i$th components; and analogously for algebra graphs.

If $\Sigma|\pi$ is a signature with products, we use $\mathbf{Alg}(\Sigma|\pi)$ to denote the category of algebras over $\Sigma|\pi$ and $\mathbf{AlgGraph}^+(\Sigma|\pi)$ [resp. $\mathbf{AlgGraph}(\Sigma|\pi)$] to denote the category of [deterministic] algebra graphs over $\Sigma|\pi$. The following extends Theorem 9 to signatures with products.

**Theorem 11.** *For any $\Sigma|\pi$,* $\mathbf{Alg}(\Sigma|\pi)$ *and* $\mathbf{AlgGraph}(\Sigma|\pi)$ *are equivalent.*

The following result states that we can indeed flatten arbitrary signatures into signatures with products, and obtain equivalent categories of algebras.

**Theorem 12.** *For any $\Sigma$, there is a signature with products* $\mathsf{flat}(\Sigma)$ *such that* $\mathbf{Alg}(\Sigma)$ *and* $\mathbf{Alg}(\mathsf{flat}(\Sigma))$ *are equivalent.*

To construct $\mathsf{flat}(\Sigma)$, we need to add product sorts and projection operators. For this purpose, assume disjoint subsets of product sort names and projection operator names, which are also disjoint from $S$ and $O$. For all $z \in S^*$, let $s_z$ denote a distinct fresh product sort name corresponding to $z$, and for all $1 \leq i \leq |z|$, let $p_{z,i}$ denote a distinct fresh projection operator name from $s_z$-values to their $i$th components. Now $\mathsf{flat}(\Sigma)$ is defined as $\Sigma_1|\pi$, where $\Sigma_1$ consists of[1]

$$
\begin{aligned}
S_1 &= S \cup \{s_{\sigma(o)} \mid o \in O\} \\
O_1 &= O \cup \{p_{\sigma(o),i} \mid o \in O, 1 \leq i \leq \alpha(o)\} \\
\sigma_1 &= \{(o, s_{\sigma(o)}) \mid o \in O\} \cup \{(p_{\sigma(o),i}, s_{\sigma(o)}) \mid o \in O, 1 \leq i \leq \alpha(o)\} \\
\tau_1 &= \tau \cup \{(p_{\sigma(o),i}, \sigma(o)|_i) \mid o \in O, 1 \leq i \leq \alpha(o)\} \\
\pi &= \{(s_{\sigma(o)}, p_{\sigma(o),1} \cdots p_{\sigma(o),\alpha(o)}) \mid o \in O\}.
\end{aligned}
$$

By combining the above results, we get

**Corollary 13.** *For any $\Sigma$,* $\mathbf{Alg}(\Sigma)$ *and* $\mathbf{AlgGraph}(\mathsf{flat}(\Sigma))$ *are equivalent.*

### 2.2 Reflected Graph Embeddings

To achieve graph attribution, we embed algebra graphs into larger graphs. To define the necessary constructs, let $\subseteq$ define the component-wise subset relation over graphs.

**Definition 14 (graph embedding).** *Let* $\mathbf{G}$ *be a sub-category of* **Graph**. *A* graph embedding *over* $\mathbf{G}$ *is a pair* $(G^-, G)$ *such that* $G^- \in \mathbf{G}$ *and* $G^- \subseteq G \in$ **Graph**. *If* $(G^-, G), (H^-, H)$ *are graph embeddings, then a* reflection *from* $(G^-, G)$ *to* $(H^-, H)$ *is a graph morphism* $h\colon G \to H$ *such that for all* $n \in N_G$, $h^N(n) \in N_{H^-}$ *implies* $n \in N_{G^-}$, *and for all* $e \in E_G$, $h^E(e) \in E_{H^-}$ *implies* $e \in E_{G^-}$. $\mathbf{REmb}(\mathbf{G})$ *denotes the category of graph embeddings over* $\mathbf{G}$ *with reflections as arrows.*

*A graph embedding* $(G^-, G)$ *is said to be* glued over *a discrete graph* $G^{--} \subseteq G^-$, *if for all* $e \in E_G \setminus E_{G^-}$ *and incident nodes* $n \in \{src(e), tgt(e)\}$, $n \in N_{G^-}$ *implies* $n \in N_{G^{--}}$. *An* embedding functor *is a functor* $\mathcal{E}\colon \mathbf{G} \to \mathbf{dGraph}$ *such that* $\mathcal{E}(G) \subseteq G$ *for all* $\mathbf{G}$*-graphs* $G$ *and* $\mathcal{E}(f) = f \restriction \mathcal{E}(G)$ *for all* $\mathbf{G}$*-morphisms* $f\colon G \to H$. $\mathbf{REmb}(\mathcal{E})$ *denotes the full sub-category of* $\mathbf{REmb}(\mathbf{G})$ *consisting of embeddings* $(G^-, G)$ *glued over* $\mathcal{E}(G^-)$.

The term *reflection* is chosen to stress that the structure of the subgraph $H^-$ is reflected (as the dual of preserved) in $G^-$.

[1] It should be noted that $\Sigma_1$ has a bipartite signature graph (and hence bipartite algebra graphs) as *every* operation is redefined to have a product sort as its source; even the operations that were already unary to start with. This is not at all necessary for the results in this paper: other constructions for $\mathsf{flat}(\Sigma)$ may be more intuitive in practice.

Thus, if an embedding $(G^-, G)$ is glued over a graph $G^{--}$, this means that only nodes in $G^{--}$ may be connected (by $G$-edges) to nodes outside $G^-$. For instance, in this paper we do not want to allow attribute edges to point to product nodes as these are meant only as auxiliaries,[2] so our embeddings will be glued over the sub-graph of the algebra graph with only non-product nodes. Very often we just use $G$ to denote graph embeddings $(G^-, G)$.

Based on this, we can define our category of attributed graphs. In this definition, $\mathcal{E}_{\Sigma|\pi}$: $\mathbf{AlgGraph}(\Sigma|\pi) \to \mathbf{dGraph}$ (for an arbitrary signature with products $\Sigma|\pi$) is the embedding functor mapping every $\Sigma|\pi$-algebra graph $G$ to the discrete sub-graph with nodes $\{n \in N_G \mid t(n) \in S_\Sigma \setminus dom(\pi)\}$, where $t$ is the typing of $G$ into $G_\Sigma$.

$$\mathbf{AttGraph}(\Sigma) = \mathbf{REmb}(\mathcal{E}_{\mathsf{flat}(\Sigma)}). \tag{1}$$

Although the formal definition may appear complicated (partially because we have set it up so that it is a special case of the general framework introduced in the next section), the basic idea is still conceptually simple: an attributed graph is a graph with an embedded deterministic algebra graph. This means that there are three types of edges in the overall graph:

- Edges within the algebra graph. These encode the algebra, as discussed above.
- Edges entirely outside the algebra graph, i.e., with end nodes also outside the algebra graph. These represent the "ordinary" graph structure.
- Edges not in the algebra graph, but with one or more end nodes in the algebra graph. These are *attribute edges*, i.e., they provide the kind of information that we introduced attributed graphs for in the first place.

*Example 15.* Figure 2 shows an example attributed graph for the signature Prim of Example 2, using the standard algebra, encoded into the graph structure. (Obviously the algebra graph is only partially shown.) Examples of algebra-only edges are the succ- and $\pi$-labelled edges; A, B and next are ordinary graph edges; and x and y are attribute edges. The italic inscriptions *0*, *1* and *true* represent the algebra values and are formally not part of the actual graph. Note that only non-product nodes are used as glue between the algebra graph to the "real" graph.

For arbitrary signatures, we first have to construct the algebra graph with product sorts; an attributed graph is then a graph with this algebra graph embedded, such that, moreover, only the non-product sorts are eligible as end nodes of the attribute edges.

With a fairly light discipline on the choice of labels, we can in fact make the definitions even easier. Namely, if we assume that operators of the signature $\Sigma$ are never used to label edges in $E_G \setminus E_{G^-}$, then $G^-$ can be constructed from $G$ by restricting to the $O$-labelled edges.

[2] This is a choice, not a necessity: one might actually want to have sorts that stand for tuples in the original, unflattened signature.

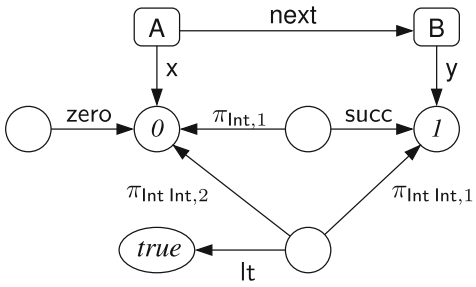



**Fig. 2.** Example attributed graph; rectangular nodes are ordinary graph nodes, ellipsoid ones represent algebra values.

We now show that this category is essentially equivalent to the standard model of [3]. We reformulate their definition so as to make the equivalence proof easier.

**Definition 16.** *Let* $\mathcal{D}\colon \mathbf{C} \to \mathbf{dGraph}$ *be a functor to discrete graphs. The category of* $\mathcal{D}$*-attributed graphs* $\mathbf{SAttGraph}(\mathcal{D})$ *is defined by*

- *Objects* $\langle G, C\rangle$ *where* $G$ *is a graph and* $C$ *an object of* $\mathbf{C}$*, such that* $\mathcal{D}(C) \subseteq G$*.*
- *Arrows* $(f\colon G \to H, g\colon B \to C)$*, where* $f$ *is a graph morphism and* $g$ *an arrow from* $\mathbf{C}$*, such that* $\mathcal{D}(g) = f \restriction \mathcal{D}(dom(g))$ *— in other words,* $f$ *and* $g$ *agree upon the discrete graph.*

Examples of functors $\mathcal{D}$ that can be "plugged in" here are:

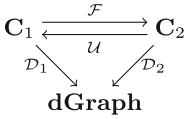
- $\mathcal{A}_\Sigma\colon \mathbf{Alg}(\Sigma) \to \mathbf{dGraph}$ for an arbitrary signature $\Sigma$, mapping every $\Sigma$-algebra $A$ to the discrete graph with $N = \bigcup_{s\in S} D^s$;
- $\mathcal{A}_{\Sigma|\pi}\colon \mathbf{Alg}(\Sigma|\pi) \to \mathbf{dGraph}$ for a signature with product sorts $\Sigma|\pi$, mapping every $\Sigma|\pi$-algebra $A$ to the discrete graph with $N = \bigcup_{s\in S\setminus dom(\pi)} D^s$;
- The functor $\mathcal{E}_{\Sigma|\pi}\colon \mathbf{AlgGraph}(\Sigma|\pi) \to \mathbf{dGraph}$ defined above.

The standard category of node-attributed graphs, as defined in [3], is essentially given by $\mathbf{SAttGraph}(\mathcal{A}_\Sigma)$ — where "essentially" means that we ignore some differences:

- In the standard model, attributed graphs are *typed*. We leave out typing because we find it complicates the presentation; moreover, enriching graphs with typing is a standard construction — see, e.g., [10].
- In the standard model, the only connections allowed between the non-attribute part of the graph and attribute (i.e., algebra) values are edges with non-data nodes as sources. We find that this constraint unnecessarily complicates the presentation and does not affect the formalism in any way; moreover, we believe that attribute edges starting in data nodes may be useful as well. Furthermore, this constraint can always be imposed on top of our definition, if so desired.

– The standard model includes *edge attributes*, which are essentially edges whose sources are edges. These present a technical complication which we have omitted, but which could be catered for by extending the category **Graph** with such edges in general.[3]

**Definition 17.** *Two functors* $\mathcal{D}_i \colon \mathbf{C}_i \to \mathbf{dGraph}$ *(*$i = 1, 2$*) are* source equivalent *if there are functors* $\mathcal{F} \colon \mathbf{C}_1 \to \mathbf{C}_2$ *and* $\mathcal{U} \colon \mathbf{C}_2 \to \mathbf{C}_1$ *which establish an equivalence between* $\mathbf{C}_1$ *and* $\mathbf{C}_2$*, and such that, moreover, the following diagram of functors commutes:*




For instance, the functors $\mathcal{A}_\Sigma$, $\mathcal{A}_{\mathsf{flat}(\Sigma)}$ and $\mathcal{E}_{\mathsf{flat}(\Sigma)}$ introduced above are pairwise source equivalent for arbitrary $\Sigma$, due to (respectively) Theorems 11 and 12.

The reason for introducing source equivalence is the following theorem, which states that replacing the "data component" in the standard model by a source equivalent one does not change the category.

**Theorem 18.** *If* $\mathcal{D}_i \colon \mathbf{C}_i \to \mathbf{dGraph}$ *for* $i = 1, 2$ *are two source equivalent functors, then* $\mathbf{SAttGraph}(\mathcal{D}_1)$ *and* $\mathbf{SAttGraph}(\mathcal{D}_2)$ *are equivalent categories.*

This is shown by functors between $\mathbf{SAttGraph}(\mathcal{D}_1)$ and $\mathbf{SAttGraph}(\mathcal{D}_2)$ that coincide with $\mathcal{D}_1$ and $\mathcal{D}_2$ on the algebra component and with the identity functor on the graph component. Note that the source equivalence precisely guarantees that the part of the algebra used in the graph remains untouched when replacing $\mathcal{D}_1$ by $\mathcal{D}_2$, and hence the identity functor can be used.

The final auxiliary result on the road to proving equivalence between the standard model and our formalisation is the following.

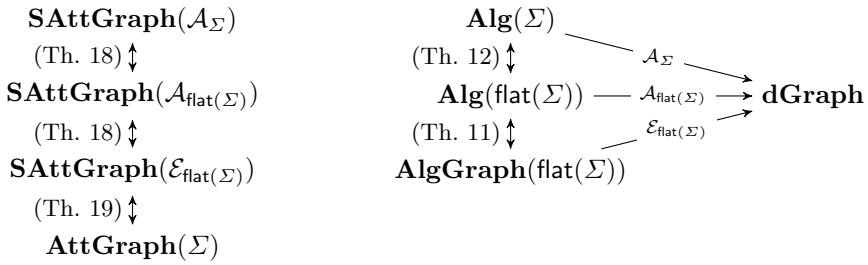
**Theorem 19.** *For any* $\Sigma|\pi$, $\mathbf{SAttGraph}(\mathcal{E}_{\Sigma|\pi})$ *and* $\mathbf{REmb}(\mathcal{E}_{\Sigma|\pi})$ *are equivalent.*

This results in the following corollary, which is the first main result of this paper:

**Corollary 20.** *For any* $\Sigma$, $\mathbf{SAttGraph}(\mathcal{A}_\Sigma)$ *and* $\mathbf{AttGraph}(\Sigma)$ *are equivalent.*

*Proof.* This follows from a chain of equivalences sketched in the following diagram.

[3] Methodologically, we believe that edge attributes are not a useful concept, since they can always be encoded by using attributed nodes instead. In a context where the increase in expressiveness is felt to be worth the price of a more complicated formalism, we believe that an extension to *hyper-edges* is typically more appropriate than edges over edges.

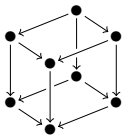



Here, $\leftrightarrow$ denotes equivalence of categories and $\rightarrow$ denotes a functor. The vertical chain on the left contains the actual steps of the proof; the diagram on the right is the justification for applying Theorem 18.

## 3 Adhesiveness

In this section we reformulate the core construction above, that of graph embeddings (Definition 14), in a more general way, getting away from the precise choice of graph category. For this, we adopt the setting of adhesive HLR categories, developed by Ehrig et al. [5] based on the adhesive categories of Lack and Sobociński [11]. One of the advantages is that, in this setting, many theorems come "for free;" an example is the *embedding theorem* used in the next section. We show that our embedding construction, generalised as the category of *reflected monos*, preserves adhesiveness, or can give rise to particular HLR adhesive categories. Among other things, this essentially constitutes an alternative proof strategy for the HLR adhesiveness of **SAttGraph**.

For lack of space, we have to omit the definitions of the basic categorical concepts. In addition we need the more involved concept of *Van Kampen squares*. A Van Kampen square in a given category is a commuting square which, if used as the bottom square in a "cube" diagram of which the back faces are pullbacks (see right), guarantees that the front faces are pullbacks if and only if the top square is a pushout.



**Definition 21 (adhesive HLR category, [5]).** *Let* $\mathbf{C}$ *be a category. A class of morphisms* $\mathcal{M}$ *in* $\mathbf{C}$ *is called* suitable *if it satisfies the following properties:*

- $\mathcal{M}$ *consists of monomorphisms;*
- $\mathcal{M}$ *is closed under isomorphisms and composition;*
- $\mathcal{M}$ *is closed under pushout and pullback.*

$\mathbf{C}$ *is called an* adhesive HLR category *for a suitable class of morphisms* $\mathcal{M}$ *if it satisfies the following properties for all* $f \in \mathcal{M}$*:*

- *Each cospan* $\bullet \xrightarrow{f} \bullet \leftarrow \bullet$ *has a pullback;*
- *Each span* $\bullet \xleftarrow{f} \bullet \rightarrow \bullet$ *has a pushout, such that the pushout diagram is a Van Kampen square.*

A category is *adhesive* in the sense of [11, Definition 5], if it is adhesive HLR for the class $\mathcal{M}$ of all monomorphisms, and moreover, all pullbacks exist. The conditions on adhesive categories essentially ensure that such categories are "set-like"; that is, the pushout is "union-like" and the pullback is "intersection-like".

For instance, our example category, **Graph**, is adhesive, as shown in [11, Proposition 8]; and so is $\mathbf{AlgGraph}^+$, due to the fact (not proved here) that $\mathbf{AlgGraph}^+$ is closed under **Graph**-pushouts and -pullbacks. On the other hand, **AlgGraph** is *not* adhesive, and indeed could not be, given that it is equivalent to **Alg** (see Theorem 9) which is well known not to be adhesive. Another observation is that in any category **C** the class of isomorphisms is suitable in the sense of Definition 21; since, moreover, pushouts and pullbacks over isomorphisms always trivially exist, the following is easy to show:

**Proposition 22.** *Every category is adhesive HLR for the class $\mathcal{M}$ of isomorphisms.*

### 3.1 Reflected Monos

We now define a categorical construction generalising reflected embeddings (Definition 14).

**Definition 23 (reflected monos).** *Let* **C** *be an arbitrary category. The category of* reflected monos *in* **C**, *denoted* $\mathbf{RMon}(\mathbf{C})$, *is defined as follows:*

- *Objects are monos $a\colon A \hookrightarrow B$ of* **C**; *we write $a^-$ and $a^+$ for the* inner object *$A$ and* outer object *$B$, respectively;*
- *Arrows $f\colon a \to b$ are pairs of arrows $(f^-\colon a^- \to b^-, f^+\colon a^+ \to b^+)$ from* **C** *such that the resulting square is a pullback diagram:*

$$\begin{array}{ccc} a^+ & \xrightarrow{f^+} & b^+ \\ {\scriptstyle a}\uparrow & PB & \uparrow{\scriptstyle b} \\ a^- & \xrightarrow{f^-} & b^- \end{array}$$

*Identities and arrow composition are defined component-wise.*

Note that this indeed gives rise to a category; in particular, arrow composition is correct due to the pullback composition property.

The intuition behind the definition of **RMon** is that monos $a$, in set-like categories, are essentially embeddings of the inner object $a^-$ into the outer object $a^+$. We will refer to the part of $a^+$ that is "disjoint" from $a^-$ as the *rim* of $a$; this may be thought of as the largest sub-object of $a^+$ which, when taking the coproduct with $a^-$, is still a sub-object of $a^+$. The pullback property of the morphisms $f\colon a \to b$ ensures that none of the rim of $a$ "spills over" into the inner object $b^-$; or in other words, $b^-$ is *reflected* in $a^-$. Some more observations:

- If **C** has an initial object 0, then monos $0 \hookrightarrow A$ have an "empty inner object"; essentially, the entire object $A$ is rim. We call such objects *closed*.

- Intuitively, the outer object $a^+$ consists of the rim, the inner object, and some additional structure connecting the inner object to the rim. We informally refer to this connecting structure as "glue." For instance, in the case of attributed graphs, the glue is the set of attribute edges.
- In general, arrows $f$ incorporate changes to both the rim, the inner object and the glue. Arrows $f$ that completely preserve the inner object are characterised by the fact that $f^-$ is an isomorphism; we call such arrows *inner isomorphisms*. Preservation of the rim, on the other hand, can be captured by requiring that the pullback diagram of $f$ is also a pushout diagram (in $\mathbf{C}$). Finally, if $\mathbf{C}$ has an initial object, then the simultaneous preservation of the inner object and the glue can also be captured; see Definition 35.
- Due to the well-definedness of pullbacks up to isomorphism, every arrow $f\colon a \to b$ in $\mathbf{RMon}$ is essentially determined by its outer component, $f^+$.

The following is another core result of this paper. To prove it, we first need to establish that monos in $\mathbf{RMon}(\mathbf{C})$ are pairs of (outer and inner) monos in $\mathbf{C}$; pushouts over monos in $\mathbf{RMon}(\mathbf{C})$ consist of outer pushouts and inner VK squares in $\mathbf{C}$; and pullbacks in $\mathbf{RMon}(\mathbf{C})$ consist of outer and inner pullbacks in $\mathbf{C}$.

**Theorem 24.** *If* $\mathbf{C}$ *is an adhesive category, then so is* $\mathbf{RMon}(\mathbf{C})$.

Unfortunately, reflected monos do not yet capture the category $\mathbf{AttGraph}(\Sigma)$ defined in (1), since for $\mathbf{AttGraph}(\Sigma)$ we had the following further constraints:

- Inner graphs were restricted to the sub-category of algebra graphs over $\mathsf{flat}(\Sigma)$;
- Embeddings were restricted to those glued over a further sub-graph.

We will show how to lift the first kind of restriction to reflected monos, and very briefly hint on how to achieve the second. For a full sub-category $\mathbf{D}$ of $\mathbf{C}$, let $\mathbf{RMon}(\mathbf{D}, \mathbf{C})$ denote the full sub-category of $\mathbf{RMon}(\mathbf{C})$ such that all inner objects are in $\mathbf{D}$.

**Proposition 25.** *For any full subcategory* $\mathbf{G}$ *of* $\mathbf{Graph}$, $\mathbf{REmb}(\mathbf{G})$ *is equivalent with* $\mathbf{RMon}(\mathbf{G}, \mathbf{Graph})$.

For example, $\mathbf{REmb}(\mathbf{AlgGraph})$ is equivalent to $\mathbf{RMon}(\mathbf{AlgGraph}, \mathbf{Graph})$. The reason why this equivalence is not an isomorphism is that there are many monos that correspond to a single graph embedding. Now let us call $\mathbf{D}$ *closed under* $\mathcal{M}$-pushouts/pullbacks where $\mathcal{M}$ is a suitable class of morphisms if, for every [co]span in $\mathbf{D}$ with one of the morphisms in $\mathcal{M}$, the corresponding $\mathbf{C}$-pushout object [$\mathbf{C}$-pullback object] is also in $\mathbf{D}$.

**Theorem 26.** *If* $\mathbf{C}$ *is an adhesive category,* $\mathbf{D}$ *is a full sub-category of* $\mathbf{C}$, $\mathcal{M}$ *is a suitable class of morphisms in* $\mathbf{D}$, *and* $\mathbf{D}$ *is closed under* $\mathcal{M}$*-pushouts/pullbacks, then* $\mathbf{D}$ *is adhesive HLR for the class* $\mathcal{M}$, *and* $\mathbf{RMon}(\mathbf{D}, \mathbf{C})$ *is adhesive HLR for the class* $\mathcal{N}$ *of all monomorphisms with inner arrow in* $\mathcal{M}$.

*Proof.* This follows from the fact that the constructions of the pushouts and pullbacks in $\mathbf{RMon}(\mathbf{C})$ entirely rely on the corresponding $\mathbf{C}$-constructions over the inner and outer parts of the objects and arrows. We have assumed $\mathbf{D}$ to be closed under these constructions, hence the resulting objects are in $\mathbf{RMon}(\mathbf{D}, \mathbf{C})$; moreover, $\mathcal{D}$ is a full sub-category, hence the constructed objects also satisfy the necessary universal properties. It follows that all required pushouts and pullbacks exist.

An application of this result is the following.

**Corollary 27.** *Let $\Sigma$ be an arbitrary signature.*

1. $\mathbf{REmb}(\mathbf{AlgGraph}^{+}(\Sigma|\pi))$ *is adhesive.*
2. $\mathbf{REmb}(\mathbf{AlgGraph}(\Sigma|\pi))$ *is adhesive HLR for inner isomorphic monos.*

To also lift the "gluing over"-construction of Definition 14 to reflected monos, instead of just a sub-category $\mathbf{D}$, we need a functor $\mathcal{E}\colon \mathbf{D} \to \mathbf{RMon}(\mathbf{E}, \mathbf{D})$, with $\mathbf{E}$ a further full sub-category of $\mathbf{D}$, such that $\mathcal{E}(G)^{+} = G$ and $\mathcal{E}(f)^{+} = f$ for all objects $G$ and arrows $f$ of $\mathbf{D}$. We can then define $\mathbf{RMon}(\mathcal{E}, \mathbf{C})$ as the full sub-category of $\mathbf{RMon}(\mathbf{D}, \mathbf{C})$ with objects $a$ such that the diagram $\bullet \overset{\epsilon(a)}{\to} \bullet \overset{a}{\to} \bullet$ has a pushout complement.

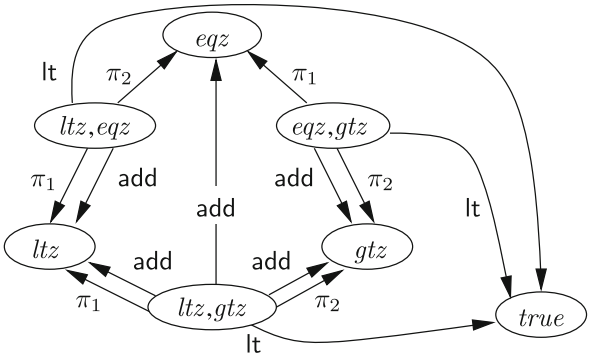



**Fig. 3.** Partial non-deterministic algebra graph for Prim of Example 2.

## 4 Data Abstraction

One of the most powerful analysis techniques for dynamic behaviour is *abstraction*. This involves discarding information from a model in order to make it more tractable, and over-approximating the original system by (where necessary) "guessing" what the discarded information may have been.

In a graph-based setting, a very natural kind of abstraction is obtained by taking a non-injective image of the start graph and applying the rules to that. The (standard) *embedding theorem* then implies that, under a certain consistency condition (Definition 30 below), all transformations on the original graph

can be applied to the abstract graph. (Other studies of abstraction for graph transformation are reported in [1,16,18].)

In this section, we show how *data abstraction*, i.e., where only the data domain and not the "proper" graph structure is abstracted, can be formulated in the framework of reflected monos. In this regard, our framework is more powerful than the standard attributed graph model, due to the ability to deal with non-determinism. The embedding theorem automatically holds due to adhesiveness; we show that this abstraction also automatically fulfills consistency, and that it is still valid in the presence of negative application conditions that only constrain the rim (i.e., the proper graph part).

*Example 28.* Figure 3 shows a partial abstract algebra graph $G$ for $\mathsf{flat}(\mathsf{Prim})$, with $\mathsf{Prim}$ as in Example 2. There is a non-injective morphism $h$ from the natural algebra graph $H$ for $\mathsf{flat}(\mathsf{Prim})$ (partially displayed in Fig. 2) to $G$, with especially, for all $i \in N_H^{\mathsf{Int}}$,

$$h : i \mapsto \begin{cases} ltz & \text{if } i < 0 \\ eqz & \text{if } i = 0 \\ gtz & \text{if } i > 0. \end{cases}$$

As can be seen from Fig. 3, $G$ is not deterministic: for instance, from the tuple element $(ltz, gtz)$ there are three outgoing $\mathsf{add}$-arrows, reflecting the fact that adding a negative to a positive number might give a negative, zero, or positive result.

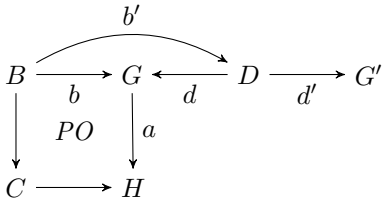
In contrast, the only non-injective algebra morphism from the natural algebra over $\mathsf{Prim}$ is to the point algebra, in which every sort has exactly one element. This abstraction loses all data distinctions and is therefore much too coarse for almost all uses.

**Definition 29 (inner abstraction morphism).** *An* inner abstraction morphism *is an arrow in* $\mathbf{RMon}(\mathbf{C})$ *that is a pushout in* $\mathbf{C}$.

As discussed in Sect. 3, an arrow in $\mathbf{RMon}(\mathbf{C})$ that is a pushout in $\mathbf{C}$ essentially does not modify the outer object — except to accommodate changes in the inner object.

To recall the embedding theorem, first we need the following *consistency condition.*

**Definition 30 (consistency, cf. [4,6,12]).** *A morphism* $a\colon G \to H$ *is called* consistent with *a span* $G \xleftarrow{d} D \xrightarrow{d'} G'$ *if a commuting diagram of the following shape exists:*



Intuitively, consistency comes down to the requirement that none of the items of $G$ that are deleted by the span (meaning that they are not in $d$-image of $D$) are "modified" by $a$ — where modification means (node or edge) merging or addition of incident edges. The embedding theorem refers to the derived span of a transformation sequence, which we will not formally define; however, in an adhesive HLR category with a class $\mathcal{M}$ of monos, the morphisms of derived spans are always in $\mathcal{M}$.

**Theorem 31 (embedding, cf. [4,6,14]).** *For any transformation $t : G_0 \stackrel{*}{\Longrightarrow} G_n$ and morphism $a_0 \colon G_0 \to H_0$ that is consistent with the derived span of $t$, there is a transformation $H_0 \stackrel{*}{\Longrightarrow} H_n$ consisting of the same rules as $t$, and a morphism $a_n \colon G_n \to H_n$.*

The following lemma implies a sufficient condition for consistency.

**Lemma 32.** *Let* **C** *be an adhesive category. If $G \stackrel{d}{\leftarrow} D \stackrel{d'}{\rightarrow} G'$ is a span of inner isomorphic monos and $a \colon G \to H$ is an inner abstraction in a category* **RMon**(**C**)*, then there is a diagram of the following shape, where $e$ and $a'$ are also inner abstractions:*

$$\begin{array}{ccccc} G & \xleftarrow{d} & D & \xrightarrow{d'} & G' \\ {\scriptstyle a}\downarrow & PO & {\scriptstyle e}\downarrow & PO & \downarrow{\scriptstyle a'} \\ H & \xleftarrow{c} & E & \xrightarrow{c'} & H' \end{array}$$

This means that, for categories where all the rule morphisms are inner isomorphic monos, inner abstractions are always consistent.

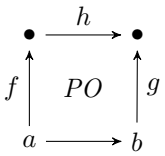
**Corollary 33 (abstraction embedding).** *Consider a sub-category of* **RMon** *which is adhesive HLR for a class $\mathcal{M}$ of inner isomorphisms. For any transformation $t : G_0 \stackrel{*}{\Longrightarrow} G_n$ and any inner abstraction $a_0 \colon G_0 \to H_0$, there is a transformation $H_0 \stackrel{*}{\Longrightarrow} H_n$ consisting of the same rules as $t$, with an inner abstraction $a_n \colon G_n \to H_n$.*

*Negative application conditions.* Negative application conditions (NACs) in combination with abstraction pose a problem: structures forbidden by a NAC may very well (appear to) exist on the abstract level, whereas they do not occur in the corresponding concrete graph. In general, to cope with this we can only "switch off" the evaluation of NACs on the abstract level; however, this makes the resulting over-approximation very coarse. The last result of this paper is to extend abstraction embedding to rules with NACs that do not constrain the inner objects. We first have to recall how NACs work.

**Definition 34 (negative application condition).** *A* negative application condition *is a morphism $n \colon L \to N$. $n$ is said to be* satisfied *by a matching $m \colon L \to G$ if $m$ does not factor through $n$, i.e., there is no $f \colon N \to G$ such that $m = f \circ n$.*

To avoid the problem of false positives after abstraction, it not enough to restrict the NACs to inner isomorphisms: they should also not introduce any new connections between the inner object and the rim. To formulate this as a general requirement, we have to assume that the base category has an initial object.

**Definition 35.** *Let* $\mathbf{C}$ *be an adhesive category with an initial object. A morphism* $h$ *in* $\mathbf{RMon}(\mathbf{C})$ *is said to* avoid the inner object *if* $h$ *is part of a pushout diagram of the following form, where* $a$ *and* $b$ *are closed objects (meaning that* $a^-$ *and* $b^-$ *are empty):*




The intuition is that a NAC avoids the inner object if it does not constrain the inner object itself, nor the glue between the inner object and the rim. If a NAC avoids the inner object, then inner abstractions do not cause false negatives.

**Theorem 36.** *Assume* $\mathbf{C}$ *is an adhesive category with an initial object; let* $n\colon L \to N$ *be a NAC in* $\mathbf{RMon}(\mathbf{C})$ *that avoids the inner object,* $m\colon L \to G$ *a matching, and* $a\colon G \to H$ *an inner abstraction. If* $m$ *satisfies* $n$*, then* $a \circ m$ *satisfies* $n$*.*

It follows that Corollary 33 continues holding for rules with NACs that avoid the inner object.

## 5 Implementation

Here we show how the ideas exposed above have been partially implemented in the tool GROOVE (see [17]). GROOVE supports a basic signature $\Sigma$ consisting of four sorts: whole (integer) numbers, floating point numbers, boolean and strings, with the typical operations found in programming languages.

As an example we take a graph transformation system that models the behaviour of an *indexed stack*, which is a stack modelled using an indexed list (rather than a linked list, as is more common for this particular data structure) for the elements. That is, elements on the stack have an *order*, which is 1 for the bottom element and increases for every next element on top of it. Figure 4 shows the graphs for an empty stack and a stack with three elements, using the natural algebra graphs for the sorts at hand (actually, in this example only integers). The node labels Stack and Cell are notational conventions for self-edges with those labels, which in practice serve as node types. The Stack-node has a length-edge to the number of elements currently contained in the stack; every Cell-node has an order-edge to its index. GROOVE supports single-pushout rules in general, but can also be restricted to double-pushout. Rules are thus spans of

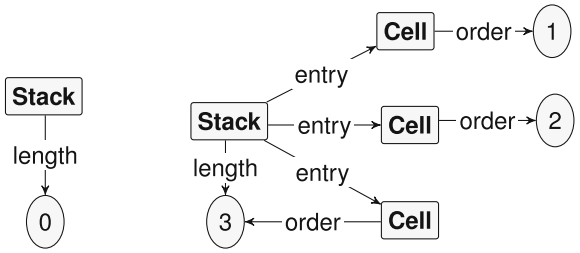



**Fig. 4.** Empty stack and 3-element stack

morphisms over rule graphs, in which the algebra subgraphs consist only of the constants from the signature and typed variable nodes for the four basic sorts; the values for the product sorts correspond to tuples of the above.

Typical operations on indexed stacks are pushing and popping elements. These are modelled by the rules shown in Fig. 5. The figure only shows the left hand side and right hand side graphs of both rules, leaving out the middle (interface) graph and suggesting the morphisms though the positioning of the nodes. For demonstration purposes, the *push* rule has been enriched with a condition that is satisfied only if the length of the stack is smaller than 5. The following graphical notational conventions are used:

- Only the relevant algebra graph nodes are shown in the figures. In particular, in host graphs, none of the auxiliary product nodes are ever included.
- Pure data nodes, i.e., elements of the data sets of the four basic sorts, are represented as ellipses labelled or by their values, by their types if they are variable nodes.
- Product nodes are represented as diamonds. The projection edges are labelled $\pi i$ for index $i$ starting at 0. The operator edges in Fig. 5 are add and lt in *push*, for addition and less-than, and sub in *pop* for subtraction.

In GROOVE, only part of the potential power of this paper's approach has been realised, in that non-deterministic algebra graphs such as the one in Fig. 3 are not supported. What is supported, on the other hand, are several (families) of algebras, namely

- *Point algebras*, where every value set consists of a single data value; i.e., all distinctions between data values are lost. If we interpret our indexed stacks under the point algebra, for instance, all order-edges point to the single integer representative, and rule *push* remains forever enabled because the lt-edge always points to the single Boolean value that represents both true and false.
- *Java algebras*, where every value set corresponds to its natural Java type, e.g., `int` for integers. This means that integer overflow is treated as Java does, by ignoring any significant bits above 31.
- *Big algebras*, where the most precise Java types available are chosen as value sets instead; e.g., `BigInteger` for integers.

– *Term algebras*, where every value set is given by the set of syntactic terms of the corresponding sort. Interpreted under the term algebra, for instance, rule *push* is not applicable to either of the graphs in Fig. 4, as in the term algebra graph the lt-edge leading from the tuple $\langle 0, 5 \rangle$ does not point to true but to the term $\mathsf{lt}(0, 5)$, which is (in that algebra) distinct from true.

## 6 Evaluation and Conclusion

In this paper we have proposed a new approach to model attributed graphs, which is more uniform than the standard model of [3] in that it stays entirely within a single (graph) category. Rather than resorting to a separate category of algebras to model the data, we encode the entire algebra structure into a sub-graph. This removes the need for additional algebraic equations specified outside the graph formalism and a corresponding satisfaction engine; thus, both tool implementers and users may benefit.

Contributions of the paper are:

– Equivalence of our model with the standard model (Corollary 20);
– An alternative proof of the adhesiveness of our construction (Theorem 26);
– Embedding theorems for data abstraction, without consistency condition (Corollary 33) and in the presence of negative application conditions (Theorem 36).

We have chosen a very common graph category in this paper: labelled binary graphs. The use of hyper-graphs instead would probably ease the encoding of the algebras. In particular, this would obviate the need for the product sorts, removing one important source of complexity. As a consequence, for instance, we would not have to flatten the signatures, and we would not have to resort to the "gluing over"-construction.

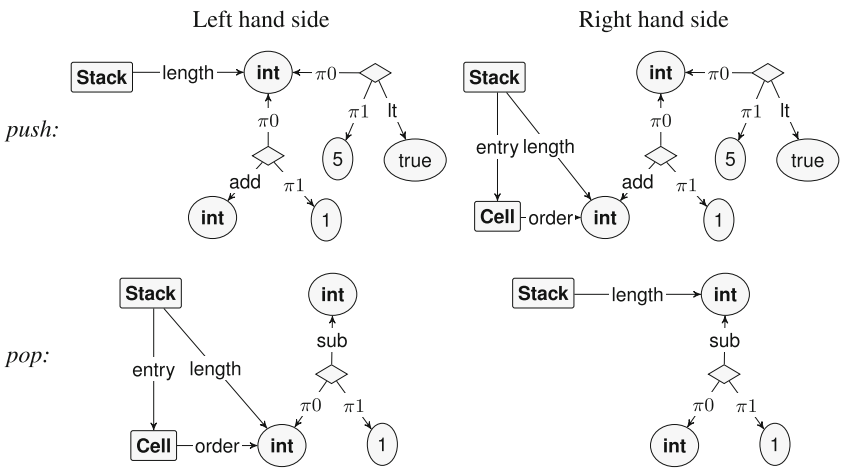



**Fig. 5.** Push and pop rules for the indexed stack

It should be noted that we have more or less silently restricted ourselves to node attributes. To support edge attributes as well, an extension of the standard notion of graph would be required in which (some) edges can have edges as their source, instead of nodes, just like in the standard model.

As we have briefly shown in Sect. 5, the setup described in this paper has been partially implemented in the tool GROOVE. It should be said, however, that the setup is not very appealing in terms of readability: for instance, already the fairly simple rules in Fig. 5 are non-trivial to read and write. In the newer versions of the tool, therefore, a lot of syntactic sugar has been added that allows the use of terms rather than product nodes, bringing it visually much closer to the standard model.

*Related work.* We have at several places referred to the "standard model" of representing attributes developed by Ehrig and al, but there are a number of other alternatives approaches. For instance, in the language *GP* for Graph Programs (e.g., [15]), attributes are encoded in labels: rules are able to compose and decompose such labels into their constituent values. In [2], the authors propose to associate exactly one attribute to every node and edge which may however be a tuple and so carry as many primitive values as one might wish. Morphisms have, apart from a structural backbone, a $\lambda$-term for each target graph element that expresses how its attribute is computed from the morphisms source. Refinements on the theme of adhesiveness that improve the way attributes fit have been studied and proposed in [6,14]. Another recent approach has been proposed in [13], using the *symbolic graphs* also studied in [12]. However, as the onderlying models are still algebras, and hence deterministic, we believe that symbolic graphs are not able to offer data abstraction in the sense of Sect. 4.

In related work of another type, an idea very similar to the one worked out in this paper has been used in [9] to extend a technique that was only available for graphs without attributes. This supports the point, made in the introduction, that there is a benefit to stick to the framework of graphs to encode the world of algebras.

**Acknowledgement.** For the proof of adhesiveness of **RMon**, we are very grateful for help from Andrea Corradini, Tobias Heindel, and Ulrike Prange.

## References

1. Bauer, J., Wilhelm, R.: Static analysis of dynamic communication systems by partner abstraction. In: Nielson, H.R., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 249–264. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74061-2_16
2. Boisvert, B., Féraud, L., Soloviev, S.: Typed lambda-terms in categorical attributed graph transformation. In: Durán, F., Rusu, V. (eds.) Algebraic Methods in Model-based Software Engineering (AMMSE). Electr. Notes Theor. Comput. Sci., vol. 56, pp. 33–47 (2011)
3. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamental theory for typed attributed graphs and graph transformation based on adhesive HLR categories. Fund. Inf. **74**(1), 31–61 (2006)

4. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-31188-2
5. Ehrig, H., Padberg, J., Prange, U., Habel, A.: Adhesive high-level replacement systems: a new categorical framework for graph transformation. Fund. Inf. **74**(1), 1–29 (2006)
6. Golas, U.: A general attribution concept for models in $\mathcal{M}$-adhesive transformation systems. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2012. LNCS, vol. 7562, pp. 187–202. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33654-6_13
7. Kastenberg, H.: Towards attributed graphs in GROOVE: Work in progress. In: Heckel, R., König, B., Rensink, A. (eds.) Graph Transformation for Verification and Concurrency (GT-VC). Electr. Proc. Theor. Comput. Sci., vol. 154, pp. 47–54 (2006)
8. Kastenberg, H., Rensink, A.: Graph attribution through sub-graphs. CTIT Technical report TR-CTIT-12-27, Department of Computer Science, University of Twente (2012)
9. Kehrer, T., Alshanqiti, A., Heckel, R.: Automatic inference of rule-based specifications of complex in-place model transformations. In: Guerra, E., van den Brand, M. (eds.) ICMT 2017. LNCS, vol. 10374, pp. 92–107. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61473-1_7
10. König, B.: A general framework for types in graph rewriting. Acta Inf. **42**(4–5), 349–388 (2005)
11. Lack, S., Sobociński, P.: Adhesive categories. In: Walukiewicz, I. (ed.) FoSSaCS 2004. LNCS, vol. 2987, pp. 273–288. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24727-2_20
12. Orejas, F.: Symbolic graphs for attributed graph constraints. J. Symb. Comput. **46**(3), 294–315 (2011)
13. Orejas, F., Lambers, L.: Symbolic attributed graphs for attributed graph transformation. In: Graph and Model Transformation (GraMoT). Electr. Comm. of the EASST., vol. 30 (2010)
14. Peuser, C., Habel, A.: Composition of $m, n$-adhesive categories with application to attribution of graphs. In: Plump, D. (ed.) Graph Computation Models (GCM). Electr. Comm. of the EASST, vol. 73 (2015)
15. Plump, D., Steinert, S.: Towards graph programs for graph algorithms. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 128–143. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30203-2_11
16. Rensink, A.: Canonical graph shapes. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 401–415. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24725-8_28
17. Rensink, A.: The GROOVE simulator: a tool for state space generation. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 479–485. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-25959-6_40
18. Rensink, A., Distefano, D.: Abstract graph transformation. Electr. Notes Theor. Comput. Sci. **157**(1), 39–59 (2006)