Electronic Communications of the EASST
Volume XXX (2013)

Proceedings of the 12th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2013)

Concatenation and other Closure Properties of Recognizable Languages in Adhesive Categories

H.J. Sander Bruggink, Barbara König, Sebastian Küpper

14 pages

Guest Editors: Matthias Tichy, Leila Ribeiro
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Concatenation and other Closure Properties of Recognizable Languages in Adhesive Categories*

H.J. Sander Bruggink[1], **Barbara König**[2], **Sebastian Küpper**[3]

[1] sander.bruggink@uni-due.de
[2] barbara_koenig@uni-due.de
[3] sebastian.kuepper@uni-due.de
Fakultät für Ingenieurwissenschaften
Abteilung für Informatik und Angewandte Kognitionswissenschaften
Universität Duisburg-Essen, Germany

Abstract: We consider recognizable languages of cospans in adhesive categories, of which recognizable graph languages are a special case. We show that such languages are closed under concatenation, i.e. under cospan composition, by providing a concrete construction that creates a concatenation automaton from two given automata. The construction is considerably more complex than the corresponding construction for finite automata. We conclude by showing negative closure properties for Kleene star and substitution.

Keywords: closure properties, recognizable languages, recognizable graph languages, concatenation, adhesive categories

1 Introduction

Regular languages for words and trees are an indispensable concept in many areas of computer science: they are used for instance for parsers, text editors or verification tools. Hence a natural question to ask is whether their counterpart in the world of graphs, recognizable graph languages à la Courcelle [Cou94], can serve the same purpose. In order to use such languages in practice it is necessary to prove closure properties and to give effective procedures in order to constructively realize such closure properties.

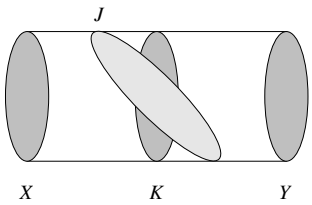
Here we focus on the closure of recognizable graph languages under concatenation. More concretely, we use an automaton model introduced by us in [BK08, BBEK12], which accepts exactly the recognizable graph languages of Courcelle (for an exact comparison see [BK08]). Such automata accept graphs equipped with a fixed inner and outer interface (categorically: cospans of graphs). Given two languages $L_{X,K}$, $L_{K,Y}$ of cospans, where the cospans are equipped with interface X,K, respectively K,Y, by concatenating every cospan of the first language with every cospan of the second language, one obtains a language $L_{X,K};L_{K,Y}$ of cospans equipped with interfaces X,Y.

For finite automata on words concatenation can be implemented by a straightforward construction [HMU06]), however for graph languages the situation is considerably more complex, due to the fact that graphs are not freely generated, as opposed to words which are the free monoid

* This work was supported by the DFG-project GaReV.

over a given alphabet. In other words: there is no canonical way to decompose a graph into atomic components (or cospans) and hence several decompositions generate the same graph. For graph automata this requires the condition that a graph is accepted independently of its specific decomposition and that two different decompositions of a cospan yield the same transitions in the automaton.

We can assume that two automata $\mathcal{A}_{X,K}, \mathcal{A}_{K,Y}$ satisfying this requirement and accepting the languages $L_{X,K}$, $L_{K,Y}$ are given. Now the challenge is to construct an automaton for the concatenation language that also adheres to this constraint. Intuitively the problem is the following: when accepting a cospan of the concatenation language, one might read it in such a way that the automaton already sees parts of the second cospan before completely processing the first. This is visualized below where cospans are represented by "tube-like" structures. Instead of reading first the cospan from X to K and then the cospan from K to Y, it is possible to start with a cospan from X to some new interface J that overlaps with K and already reaches into the second cospan, without fully containing the first.



This effect has to be taken into account in the construction of the new automaton. Especially we have to record states of both automata (since we can be in both automata at the same time) and we have to record the current union of the interface graphs K and J (this union will be denoted by U) and the parts of U that are the current interfaces for the two automata.

This construction will not only be given for graph automata, but more generally for languages of cospans of (monic) arrows in adhesive categories [LS05]. The properties of adhesive categories enable us to work in distributive subobject lattices, which are the key to the construction and the proof of its correctness. This greater generality allows us to prove the result for several classes of graph-like structures (directed graphs, hypergraphs, attributed graphs, etc.).

Concatenation for recognizable graph languages was already studied by Courcelle in [Cou94], but for a different setting where graphs have only one interface and are glued over their joint interface. Furthermore Courcelle's results applies only to concrete graphs, whereas ours is shown in a general categorical setting.

We conclude the paper by giving counterexamples for closure properties that fail to hold for graph languages, while being true for word languages: closure under Kleene star and closure under substitution.

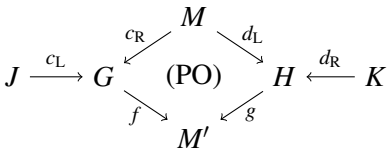
2 Recognizable Languages and Automata

2.1 Category Theory

We presuppose basic knowledge of category theory. The identity arrow of an object G will be denoted by id_G. If f and g are composable arrows, we write $f;g$ for the morphism f postcomposed

with g, i.e. $f;g = g \circ f$.

Let $\mathcal{C}$ be a category in which all pushouts exist. A cospan c in $\mathcal{C}$ is a pair of arrows $\langle c_L, c_R \rangle$ with the same codomain: $c : J \xrightarrow{c_L} G \xleftarrow{c_R} K$. J is the domain of c and K is the codomain of c. We often call J the inner interface of c and K the outer interface of c. Let $c : J \to F \leftarrow M$ and $d : M \to H \leftarrow K$ be cospans where the codomain of c equals the domain of d. Then, the composition of c and d, written as $c;d : J \to M' \leftarrow K$ is defined via the following commuting diagram where the middle diamond is a pushout



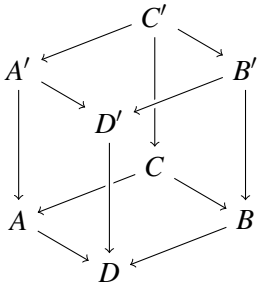
Two cospans $c\colon J \xrightarrow{c_L} G \xleftarrow{c_R} K$ and $d : J \xrightarrow{d_L} H \xleftarrow{d_R} K$ are isomorphic, written $c \sim d$, if there exists an isomorphism k from G to H, such that $c_L;k = d_L$ and $c_R;k = d_R$. A semi-abstract cospan is a $\sim$-equivalence class of cospans. We will often identify a $\sim$-equivalence class with one of its representatives. The category $\mathcal{C}ospan(\mathcal{C})$ is defined as the category that has the objects of $\mathcal{C}$ as objects and semi-abstract cospans as arrows. The identity arrows are given as the equivalence class of identity cospans $G \xrightarrow{id_G} G \xleftarrow{id_G} G$.

2.2 Adhesive Categories

For this paper we want to investigate a specific type of categories, so called adhesive categories, defined as in [LS05].

Definition 2.1. *A category $\mathcal{C}$ is called adhesive if*

- *$\mathcal{C}$ has pullbacks*
- *$\mathcal{C}$ has pushouts along monomorphisms*
- *pushouts along monomorphims are Van Kampen-pushouts, i.e. whenever the lower square in the following picture is a pushout along a monomorphim and the front and the left square are pullbacks it holds that the top square is a pushout iff the right and back squares are pullbacks.*



Definition 2.2. *An example of an adhesive category which we will use in examples is the category $\mathcal{HG}raph$. For a set A we call A^* the set of sequences of elements from A and for a function $f\colon A \to B$ we write $f^*\colon A^* \to B^*$ for the function that acts as f on each element of a sequence.*

$\mathcal{HG}raph$ has hypergraphs as objects and graph morphisms as arrows. Let Λ be a fixed set of labels with an arity function $ar\colon \Lambda \to \mathbb{N}_0$. A hypergraph is a four-tuple $G = (V_G, E_G, att_G, lab_G)$ where V_G is a set (the set of vertices), E_G is a set (the set of edges), $att_G\colon E_G \to V_G^*$ and $lab_G : E_G \to \Lambda$ are functions, where for each edge $e \in E_G$ it holds that $ar(lab_G(e)) = |att_G(e)|$. We only consider finite graphs where the set of vertices and the set of edges is finite.

A graph morphism is a pair of functions $(f_V, f_E)\colon G \to H$, $f_V\colon V_G \to V_H$, $f_E\colon E_G \to E_H$ where $lab_H \circ f_E = lab_G$ and $att_H \circ f_E = f_V^* \circ att_G$.

We say a graph D is discrete if $E_D = \emptyset$.

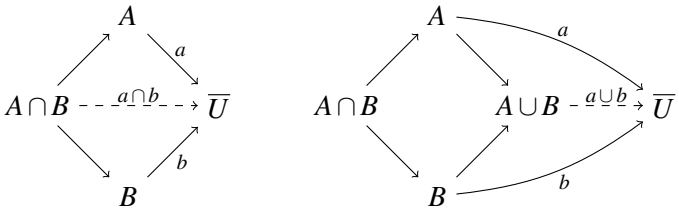
We are particularly interested in subobject lattices in adhesive categories, defined as in [LS05, BBC+11].

Definition 2.3 (Subobject)**.** *Let $\overline{U}$ be an object in $\mathcal{C}$. Two monomorphisms $a\colon A \rightarrowtail \overline{U}$ and $b\colon B \rightarrowtail \overline{U}$ are called isomorphic if there is an isomorphism $\psi\colon A \to B$ such that $\psi ; b = a$. A subobject of $\overline{U}$ is an isomorphism class of monomorphisms into $\overline{U}$. It is denoted by $[a\colon A \rightarrowtail \overline{U}]$ or $[a]$ where $a\colon A \rightarrowtail \overline{U}$ is any representative.*

For a fixed object $\overline{U}$ we consider the category $Sub(\overline{U})$ that has subobjects of $\overline{U}$ as objects and that has an arrow between two objects $[a], [b]$ if there exists an arrow c in $\mathcal{C}$ such that $c ; b = a$. In this case we write $[a] \subseteq [b]$.

An important result regarding subobject lattices due to [LS05] is that each $Sub(\overline{U})$ with partial order $\subseteq$ forms a distributive lattice, the so-called subobject lattice of $\overline{U}$.

The meet of two subobjects $[a]$, $[b]$ is realized by taking their pullback in $\mathcal{C}$ (see diagram on the left below) and is denoted by $[a] \cap [b] = [a \cap b]$. A join $[a] \cup [b] = [a \cup b]$ is obtained by taking the pushout over their meet (see diagram on the right below).



Distributivity means that $([a] \cup [b]) \cap [c] = ([a] \cap [c]) \cup ([b] \cap [c])$ and $([a] \cap [b]) \cup [c] = ([a] \cup [c]) \cap ([b] \cup [c])$ for subobjects $[a], [b], [c]$.

In order to structure our results more clearly, we define:

Definition 2.4. *Let $d\colon A \to B \leftarrow C$ be a cospan consisting of monos and $\overline{U}$ be an object with monos $a\colon A \to \overline{U}$, $b\colon B \to \overline{U}$, $c\colon C \to \overline{U}$, such that a, b, c commute with the two legs of the cospan. That is, $[a], [b], [c]$ are subobjects of $\overline{U}$ and $[a] \subseteq [b]$, $[c] \subseteq [b]$.*

In the following we write $[\widehat{d}]_{\overline{U}} = ([a], [b], [c])$ in order to represent a cospan as a triple of subobjects. When $\overline{U}$ is clear from the context, we omit it and just write $[\widehat{d}] = ([a], [b], [c])$. Union and intersection of such triples are defined pairwise.

2.3 Recognizability

Courcelle [Cou90, Cou94] introduced the notion of recognizability of graph languages as an analogon to regularity of word languages.

This notion can be extended to arbitrary categories $\mathcal{C}$ that are *locally small*, i.e. for every two objects the class of arrows between them is a set. Nondeterministic finite automata are commonly used to investigate properties of regular languages. An analogous notion for recognizable sets of arrows in a category is given by an automaton functor [BK08].

The intuition behind such automaton functors is the following: a transition function $\delta\colon Z\times\Sigma\to\mathcal{P}(Z)$ (where Z is the set of states and Σ is an alphabet) for non-deterministic word automata can be extended to a transition function $\hat{\delta}\colon Z\times\Sigma^*\to\mathcal{P}(Z)$ on words. With some currying and rearranging we can view $\hat{\delta}$ as a function that maps a word from Σ^* to a relation on Z and which furthermore satisfies $\hat{\delta}(\varepsilon)=id_Z$ and $\hat{\delta}(w_1w_2)=\hat{\delta}(w_1)\hat{\delta}(w_2)$ (functoriality conditions).

Similarly we define an automaton as a mapping from cospans of graphs to relations on states, such that the functoriality conditions above hold. Alternatively one could define automata only on so-called atomic cospans as in [BBEK12], but – different from word languages – it is necessary to require that a cospan generates the same relation, independently of its decomposition, which is already implicit in the functoriality conditions.

Definition 2.5 (Automaton)**.** *Let $\mathcal{C}$ be any category and let $\mathcal{R}el$ be the category of sets and relations. Furthermore let J,K be objects of $\mathcal{C}$.*

*A (J,K)-*automaton *is a tuple $\mathcal{A}=(A,I,F)$, where A is a functor $A\colon\mathcal{C}\to\mathcal{R}el$ that maps every object X of $\mathcal{C}$ to a finite set $A(X)$ (called the set of* states *of X) and every arrow $f\colon X\to Y$ to a relation $A(f)\subseteq A(X)\times A(Y)$. Furthermore there is a set of initial states $I\subseteq A(J)$ and a set of final states $F\subseteq A(K)$. The functor A is called* automaton functor*; we often identify $\mathcal{A}$ with its functor A.*

An automaton $\mathcal{A}$ is deterministic *whenever every relation $A(f)$ is a function and I contains exactly one element.*

The language *$L(\mathcal{A})$ of $\mathcal{A}$ (which contains arrows from J to K) is defined as follows:*

> *$f\colon J\to K$ is contained in $L(\mathcal{A})$ if and only if there exist $s\in I$, $t\in F$ which are related by $A(f)$, i.e., $s\,A(f)\,t$.*

A language $L_{J,K}$ of arrows from J to K is recognizable in $\mathcal{C}$ *if it is the language of a (J,K)-automaton.*

Many properties of regular languages can be shown for recognizable languages in a straightforward way; see [BK08] for proofs.

Proposition 2.6. *For every automaton, there exists an equivalent deterministic automaton.*

Proof. (Sketch.) The construction is more or less equivalent to the case of finite automata: we replace every set of states by its powerset. □

Proposition 2.7. *Suppose we have two recognizable languages of arrows, $L^1_{J,K}$ and $L^2_{J,K}$. Then also $L^1_{J,K}\cap L^2_{J,K}$, $L^1_{J,K}\cup L^2_{J,K}$ and $(L^1_{J,K})^{\mathrm{C}}$ (the complement of $L^1_{J,K}$) are recognizable.*

Proof. (Sketch.) Again the construction resembles the case of finite automata: For union and intersection, we take the cross product of two deterministic automata and define the final states accordingly. For the complement we exchange final and non-final states of a deterministic automaton. □

3 Closure under Concatenation

From now on we consider as category, over which to define automaton functors, the category of cospans of monos of a given adhesive category.

We want to prove that recognizable languages over adhesive categories are closed under concatenation. Let $L_{X,K}$ be a language of cospans from X to K and $L_{K,Y}$ a language of cospans from K to Y. Then

$$L_{X,K} \,;\, L_{K,Y} = \{c_1 \,;\, c_2 \mid c_1 \in L_{X,K}, c_2 \in L_{K,Y}\},$$

consists of all possible compositions of cospans of the two languages.

For regular (word) languages this closure property is well known and given two (non deterministic finite) automata $\mathcal{A}_1$ and $\mathcal{A}_2$ it is easy to construct an automaton that accepts exactly $L(\mathcal{A}_1) \,;\, L(\mathcal{A}_2)$ by first simulating $\mathcal{A}_1$ for the first part of a given word, non-deterministically switching to $\mathcal{A}_2$ and then simulating $\mathcal{A}_2$ for the rest of the word. For recognizable languages on adhesive categories we cannot retain this easy construction because an automaton functor has to behave identically on different decompositions of the same cospan, as we will show in the following example.

Example 3.1. *We concatenate two rather similar graph automata over the label set $\Lambda = \{a,b\}$. Automaton 1 accepts a cospan $J \to G \leftarrow K$ with discrete interfaces of size one of the category $\mathcal{HG}raph$. The graph G must contain an even number of a-labelled edges, no b-edges and one vertex. Automaton 2 accepts a cospan $J \to G \leftarrow K$ with discrete interfaces of size one of the category $\mathcal{HG}raph$. The graph G must contain an even number of b-labelled edges, no a-edges and one vertex. We first specify both automata. As they do essentially the same, we will only specify $\mathcal{A}_1$. We will use the notation $\#_a(G)$ for the number of a-labelled edges in a graph G.*

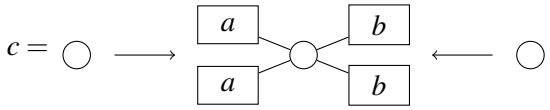
The first automaton is a 3-tuple $\mathcal{A}_1 = (A_1, I_1, F_1)$ with

$$A_1(G) = \begin{cases} \{0,1\} & \text{if } |V_G| = 1 \text{ and } |E_G| = 0 \\ \emptyset & \text{otherwise} \end{cases}$$

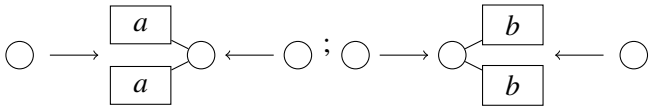
$$A_1(J \to G \leftarrow K) = \begin{cases} \{(0,0),(1,1)\} & \text{if } \#_a(G) \equiv 0 \pmod 2,\ \#_b(G) = 0 \text{ and } |V_G| = 1 \\ \{(0,1),(1,0)\} & \text{if } \#_a(G) \equiv 1 \pmod 2,\ \#_b(G) = 0 \text{ and } |V_G| = 1 \\ \emptyset & \text{otherwise} \end{cases}$$

for each cospan of hypergraphs with discrete interfaces of size one. For cospans $J \to G \leftarrow K$ that do not have discrete interfaces of size 1, $A_1(J \to G \leftarrow K) = \emptyset$. Furthermore, $I_1 = \{0\}$ and $F_1 = \{0\}$. The second automaton $\mathcal{A}_2 = (A_2, I_2, F_2)$ is constructed analogously.

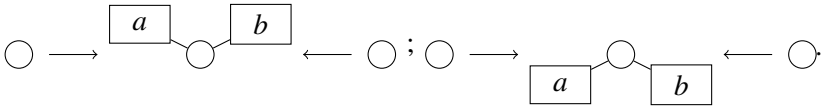
To show why we have to use a more complex construction than in the case of regular word languages, consider the following cospan:



This cospan is in the concatenation language because it can be decomposed into



and the right cospan is clearly accepted by $\mathcal{A}_1$, while the right cospan is accepted by $\mathcal{A}_2$. However, it is not possible in general to run the two automata one after the other. The same cospan c can also be decomposed in

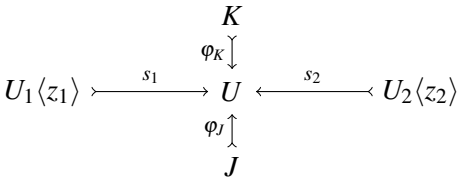


Because of the functoriality condition, the concatenation automaton must also accept the cospan in this case, although none of the two constituents cospans are accepted by $\mathcal{A}_1$ or $\mathcal{A}_2$. In general, $\mathcal{A}_1$ and $\mathcal{A}_2$ have to be run in parallel.

We will sketch a constructive proof that recognizable languages on adhesive categories are closed under concatenation, some technical details are omitted, though. We will start by defining a concatenation automaton.

Definition 3.2 (Concatenation automaton)**.** *Let $\mathcal{A}_1 = (A_1, I_1, F_1)$ and $\mathcal{A}_2 = (A_2, I_2, F_2)$ be automata over cospans of monos of an adhesive category $\mathcal{C}$ where $\mathcal{A}_1$ is an (X,K)-automaton and $\mathcal{A}_2$ is a (K,Y)-automaton. We define the concatenation automaton $\mathcal{A} = (A, I, F)$ of $\mathcal{A}_1$ and $\mathcal{A}_2$ as follows.*

- *$\mathcal{A}$ assigns to each object J a set of states $\mathcal{A}(J)$, where each state consists of four monos as shown below, and two states $z_1 \in \mathcal{A}_1(U_1), z_2 \in \mathcal{A}_2(U_2)$.*[1]

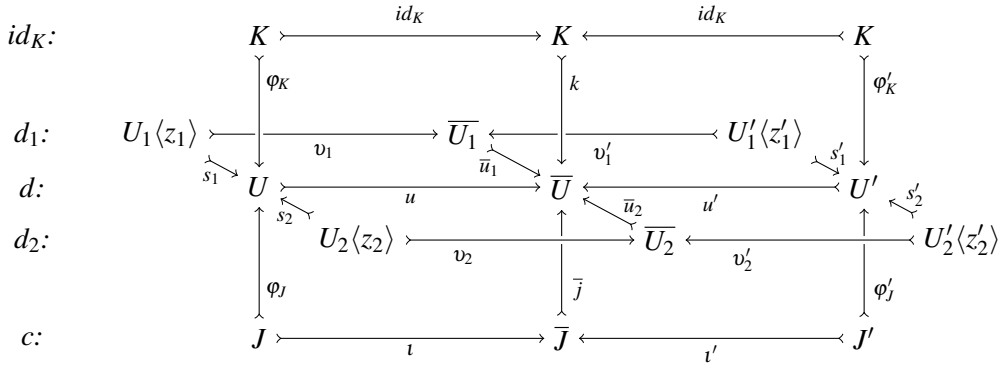


  In addition we require that $[s_1] \cup [s_2] = [id_U]$ as well as $[\varphi_J] \cup [\varphi_K] = [id_U]$ and that $[\varphi_J] \cap [\varphi_K] = [s_1] \cap [s_2]$. We want to simulate $\mathcal{A}_1$ and $\mathcal{A}_2$, so in such a state, U represents the union of the current interface J of $\mathcal{A}$ and the final interface K of $\mathcal{A}_1$. Furthermore U_1 represents the current interface of $\mathcal{A}_1$ and U_2 represents the current interface of $\mathcal{A}_2$.[2]

[1] We will indicate the two states within the categorical diagrams in angular brackets.

[2] In order to obtain a finite set of states we do not take all monos into U, but only one representative of every subobject equivalence class.

- *A state z as above is* initial *($z \in I$) if $z_1 \in I_1$, $z_2 \in I_2$, $s_1 = \varphi_J$ and $s_2 = \varphi_K$.*
- *Analogously, a state z as above is* final *($z \in F$) if $z_1 \in F_1$, $z_2 \in F_2$, $s_1 = \varphi_K$ and $s_2 = \varphi_J$.*
- *On cospans the functor $\mathcal{A}$ is defined as follows: let $c\colon J \rightarrowtail \overline{J} \leftarrowtail J'$ be a cospan consisting of two monos. Two states $(s_1, s_2, \varphi_J, \varphi_K, z_1, z_2)$, $(s_1', s_2', \varphi_J', \varphi_K', z_1', z_2')$ are related by $\mathcal{A}(c)$ whenever there are cospans d_1, d_2, d (consisting of monos) and additional monos such that the diagram below commutes*



and the following five conditions are satisfied.

1. $[\widehat{d_1}] \cup [\widehat{d_2}] = [d]$
2. $[\widehat{d_1}] \cap [\widehat{d_2}] = [\widehat{c}] \cap [\widehat{id_K}]$
3. $[\widehat{c}] \cup [\widehat{id_K}] = [d]$
4. *It holds that* $z_1\, \mathcal{A}_1(d_1)\, z_1'$ *and* $z_2\, \mathcal{A}_2(d_2)\, z_2'$.
5. $[\overline{u}_1] \cap [k] = [u_1'] \cap [k]$ *and* $[\overline{u}_2] \cap [k] = [u_2] \cap [k]$ *where* $u_2 = \upsilon_2 ; \overline{u}_2$ *and* $u_1' = \upsilon_1' ; \overline{u}_1$

The meaning of the five objects in the diagram above describing a state can be intuitively explained with the diagram depicted in the introduction (Section 1): the objects K, J from the definition above are indicated in that picture and U would be the union of K and J. If we intersect U with the cospan from X to K we obtain U_1 (fully contained in the first cospan) and if we intersect U with the cospan from K to Y we obtain U_2 (fully contained in the second cospan).

Conditions 1–3 above use the notation of Definition 2.4 and extend the conditions a state has to satisfy to transitions. The fourth condition means that d_1 is a cospan that allows a transition from z_1 to z_1' in $\mathcal{A}_1$ and that d_2 is a cospan that allows a transition from z_2 to z_2' in $\mathcal{A}_2$. Therefore, this condition ensures that the concatenation automaton behaves like $\mathcal{A}_1$ on U_1 and like $\mathcal{A}_2$ on U_2. The last condition is in place to guarantee that U_1 can only grow regarding K whereas U_2 can only diminish regarding K. Intuitively, we do not want $\mathcal{A}_1$ to forget a part of K it has already read and we do not want $\mathcal{A}_2$ to read a part of K anew that it has already forgotten.

To show that the concatenation automaton is indeed an automaton functor, we have to prove two lemmas. For the remainder of this section we assume that $\mathcal{C}$ is an adhesive category and that each cospan is a cospan consisting of monos in this category.

Lemma 3.3. *Let $\mathcal{A} = (A, I, F)$ be a concatenation automaton over $\mathcal{C}$ and c be a cospan that can be decomposed as $c = c_1 ; c_2$.*

1. *If there are states $z, \overline{z}, z'$ such that $z\, A(c_1)\, \overline{z}$ and $\overline{z}\, A(c_2)\, z'$ then a transition from z to z' via c must exist as well, that is $z\, A(c)\, z'$.*
2. *If $z\, A(c)\, z'$ for states z, z', then there is a state $\overline{z}$ such that $z\, A(c_1)\, \overline{z}$ and $\overline{z}\, A(c_2)\, z'$, that is, one can reach every state that is reachable via c by first making a c_1-step and then making a c_2-step.*

The proof to this lemma is rather technical, although it mainly uses computations of unions and intersections, and lengthy, so it is omitted here. To allow working just on the subobject-lattice of $\overline{U}$ when Lemma 3.2, we used the following supplemental result.

Lemma 3.4. *Let $a\colon A \to \overline{U}$, $b\colon B \to \overline{U}$, $c\colon C \to \overline{U}$, $d\colon D \to \overline{U}$ $a'\colon A \to \overline{U}_1$, $b'\colon B \to \overline{U}_1$, $c'\colon C \to \overline{U}_1$, $d'\colon D \to \overline{U}_1$ and $\overline{u}\colon \overline{U}_1 \to \overline{U}$ be monomorphisms in an adhesive category. Furthermore, let $a = a' ; \overline{u}$, $b = b' ; \overline{u}$, $c = c' ; \overline{u}$ and $d = d' ; \overline{u}$. Then*

(i) $[a] \cap [b] = [c] \cap [d]$ *iff* $[a'] \cap [b'] = [c'] \cap [d']$

(ii) $[a] \cup [b] = [c] \cup [d]$ *iff* $[a'] \cup [b'] = [c'] \cup [d']$.

Since the functors of concatenation automata also preserve identities, we thus conclude:

Corollary 3.5. *Concatenation automata are automaton functors.*

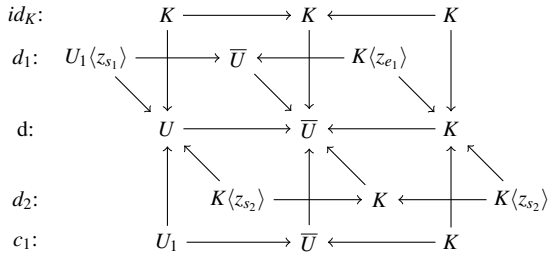
Next we want to show that concatenation automata accept exactly the language we expect them to accept.

Lemma 3.6. *Let $\mathcal{A}_1$ be an (X, K)-automaton, let $\mathcal{A}_2$ be a (K, Y)-automaton, let $\mathcal{A}$ be the concatenation automaton of $\mathcal{A}_1$ and $\mathcal{A}_2$ and let c be a cospan from X to Y.*

1. *Whenever $c \in L(\mathcal{A}_1) ; L(\mathcal{A}_2)$ then $c \in L(\mathcal{A})$.*
2. *Whenever $c \in L(\mathcal{A})$ then $c \in L(\mathcal{A}_1) ; L(\mathcal{A}_2)$.*

Proof. Part 1: Let $c \in L(\mathcal{A}_1) ; L(\mathcal{A}_2)$, then there must be a $c_1 \in L(\mathcal{A}_1)$ and a $c_2 \in L(\mathcal{A}_2)$ such that $c = c_1 ; c_2$. We will now give a transition for c_1 in $\mathcal{A}$ that starts in an initial state, then a composable transition for c_2 in $\mathcal{A}$ that ends in a final state. As we have already shown that $\mathcal{A}$ is a graph automaton, we have shown that c is accepted by $\mathcal{A}$.

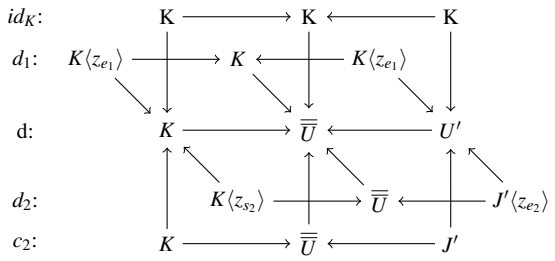
The cospan c_1 can be written as $c_1 = U_1 \to \overline{U} \leftarrow K$, hence we can construct the transition depicted in the following diagram,



where z_{s_1} is the initial and z_{e_1} the final state of the c_1-transition in $\mathcal{A}_1$, z_{s_2} is the initial and z_{e_2} the final state of the c_2-transition in $\mathcal{A}_2$. Apart from U, all objects and corresponding arrows are already known, U and its arrows are defined by first taking the pullback of $U_1 \to \overline{U}$ and $K \to \overline{U}$ and then taking the pushout of the resulting arrows, so it is the join of $U_1 \to \overline{U}$ and $K \to \overline{U}$. As before, for each object A the equivalence class of arrows going from A to $\overline{U}$ is denoted by $[a]$. We now have to show that the five properties of state transitions in $\mathcal{A}$ hold.

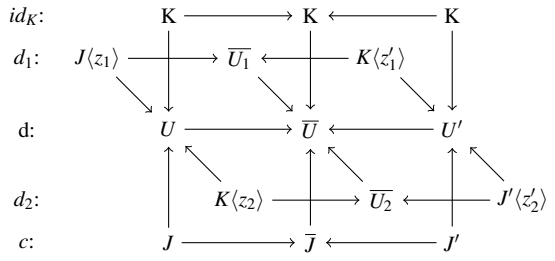
1. $[\overline{u}] = [\overline{u}] \cup [k]$, $[k] \cup [k] = [k]$ are obvious, $[u]$ is defined as the join of $[u_1]$ and $[k]$, so $[u] = [u_1] \cup [k]$ holds per definition.
2. $[k] \cap [k] = [k] \cap [k]$, $[\overline{u}] \cap [k] = [\overline{u}] \cap [k]$ and $[k] \cap [u_1] = [k] \cap [u_1]$ clearly hold.
3. Once again, $[u] = [k] \cup [u_1]$ holds per definition and $[\overline{u}] \cup [k] = [\overline{u}]$ as well as $[k] \cup [k] = [k]$ are obvious.
4. d_1 was assumed to be a cospan with $z_{s_1}\, \mathcal{A}_1\, (c_1) z_{e_1}$. As d_2 is just the identity and does not change the state, it has to be a cospan with $z_{s_1}\, \mathcal{A}_2(d_2)\, z_{s_1}$.
5. $[\overline{u}] \cap [k] = [k] \cap [k]$ and $[k] \cap [k] = [k] \cap [k]$ hold obviously.

A transition for c_2 in $\mathcal{A}$ can be found analogously:



Apart from U', all objects and corresponding arrows are known, U' can be obtained as the join of J' and K in the same way as U was constructed for the c_1-transition. The five properties are easily proven in the same way as for the c_1-transition.

Part 2: Let $c \in L(\mathcal{A})$ and an accepting c-transition in $\mathcal{A}$ be given as follows:



We will show that $c = d_1 ; d_2$. As c is in $L(\mathcal{A})$, d_1 and d_2 must be in $L(\mathcal{A}_1)$ respectively $L(\mathcal{A}_2)$, by the definition of $\mathcal{A}$, hence this is sufficient to show that $c \in L(\mathcal{A}_1) ; L(\mathcal{A}_2)$. We first observe that $[j] = [\overline{u}]$, because from d_2 it follows that $[k] \subseteq [\overline{u}_2]$ and from d_1 it follows that $[k] \subseteq [\overline{u}_1]$ and as $[\overline{u}_1] \cap [\overline{u}_2] = [j] \cap [k]$ also $[k] \subseteq [j]$ holds.

We will now continue to show that d_1 and d_2 are in fact composable over $\overline{U}$, which means that the pushout can be obtained as the union of $\overline{U_1}$ and $\overline{U_2}$. The outer interface of d_1 equals the inner interface of d_2 and as we have just seen, $[k] \subseteq [\overline{u}_1] \cap [\overline{u}_2]$ holds. Moreover, $[\overline{u}_1] \cap [\overline{u}_2] = [\overline{j}] \cap [k] \subseteq [k]$, so $[\overline{u}_1] \cap [\overline{u}_2] = [k]$ and therefore the cospans are indeed composable in this sense.

To conclude, we have to show that c and $d_1 ; d_2$ are equal. From the conditions a state transition in $\mathcal{A}$ has to fulfill, we know that $[\overline{u}_1] \cup [\overline{u}_2] = [\overline{u}]$ and thus $[\overline{u}_1] \cup [\overline{u}_2] = [\overline{j}]$, hence $c = d_1 ; d_2$. □

And thus we conclude:

Corollary 3.7. *A concatenation automaton accepts exactly the concatenation of the languages of its constituent automata.*

So we have shown:

Theorem 3.8. *Recognizable languages in adhesive categories are closed under concatenation and an automaton accepting the concatenation of two recognizable languages can be computed from their respective automata via the concatenation automaton.*

An important adhesive category is the category $\mathcal{HG}raph$ so we have particularily shown that the concatenation of recognizable graph languages is recognizable. A similar result for recognizable graph languages has been shown in [Cou94], for languages of graphs with a single interface. Note that here we also generalized this result to the setting of adhesive categories.

Example 3.9. *We give a short example of our construction. For this, we will reuse the automata from example 3.1.*

We want to construct the concatenation automaton $\mathcal{A} = (A, I, F)$ of $\mathcal{A}_1 = (A_1, I_1, F_1)$ and $\mathcal{A}_2 = (A_2, I_2, F_2)$. In the following we will use the names $c, d, J, U, \ldots$ as in Definition 3.2.

We start by defining the states of the automaton: $K = D_1$, the discrete graph with one node, because the right interface of a cospan accepted by $\mathcal{A}_1$ always has one vertex. The equality $J = U = U_1 = U_2$ has to hold, as we are working with discrete interfaces and neither $\mathcal{A}_1$ nor $\mathcal{A}_2$ accept a graph that has more or less than one vertex. Furthermore we set $\varphi_J = \varphi_K = s_1 = s_2 = id_U$. For each combination of $z_1 \in \{0,1\}$ and $z_2 \in \{0,1\}$ this forms a state. A state is initial whenever $z_0 = z_1 = 0$ and a state is final whenever $z_0 = z_1 = 0$.

So now we can define the state transitions of $\mathcal{A}$. For a given cospan c we have to define z_1, z_1', z_2, z_2' as well as d_1, d, d_2, id_K is already uniquely defined. Let $c = d$ and $c = c_1 ; c_2$ then $d_1 = c_1$, $d_2 = c_2$, $z_1 \, A_1(c_1) \, z_1'$, $z_2 \, A_2(c_2) \, z_2'$ yields a transtion in $\mathcal{A}$ and all transitions in $\mathcal{A}$ arise in this way. Looking at the states, one can see that the concatenation automaton reads a given cospan in such a way that the a-edges are counted separately in z_1 and the b-edges are counted in z_2 (both modulo 2). The resulting automaton accepts exactly the cospans of hypergraphs that have discrete interfaces of size one, exactly one vertex and an even number of a-edges, as well as an even number of b-edges.

4 Negative Results for Closure Properties

As shown above recognizable languages over adhesive categories are closed under concatenation, just like regular languages. We already know that recognizable languages are closed under

complementation, union and intersection. Unfortunately, there are some closure properties that regular languages enjoy but which do not hold for recognizable languages. To show two negative results, we will work in the category of cospans over $\mathcal{HG}raph$.

We start by showing that recognizable languages are not closed under Kleene star and we will start by defining the notion of Kleene star in our setting.

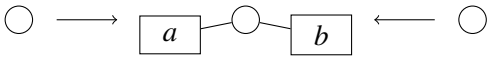
Definition 4.1. *Let L be a (K,K)-graph language. Then the Kleene-star of L, written L^*, is the (K,K)-graph language defined as:*

$$L^* = \{\sigma \mid \text{ there are } \sigma_1,\ldots,\sigma_n \text{ such that } \sigma = \sigma_1 ; \cdots ; \sigma_n \text{ and } \sigma_i \in L \text{ for all } 1 \leq i \leq n\}.$$

Hence in our setting we call the language of all cospans of graphs that can be decomposed into cospans that are in a language L, the language L^*. We will now show that recognizable graph languages are not closed under Kleene star.

Theorem 4.2. *Recognizable graph languages are not closed under Kleene-star.*

Proof. Let L be the graph language that contains only the following cospan:



Since this language consists of only one graph it is recognizable. However, L^* consists of all cospans of graphs with one vertex (which is both in the inner and outer interface), any number of a-edges and the same amount of b-edges. But we cannot find a graph automaton over the set of atomic cospans that accepts L^*, because the set of reachable states has to be the same for any decomposition of a given cospan in L^*. Thus, it is possible to decompose an accepted cospan in a way that first reads all a-edges and then all b-edges. Therefore there have to be infinitely many equivalence classes for an interface of one vertex. □

We will now see that recognizable graph languages are not closed under edge substitution (i.e., simultaneous substitution of all edges with a certain label, as in hyperedge replacement [Hab92]) either.

Definition 4.3. *Let L be a set of graphs over the label alphabet Λ, $a \in \Lambda$ a label for edges of arity k and G be a fixed graph where a sequence s of k vertices are marked as merge-vertices. The set L' of all graphs that are generated by substituting* all *a-labelled edges by G is the set of graphs that includes all graphs that can be generated from any graph $G' \in L$ by repeating the following steps until there is no more original a-edge in G'.*

- *Let e be an a-edge in G'. Delete e from G' and remember the sequence of vertices $att_{G'}(e)$.*
- *Insert a copy of G into G'. Identify the merge-vertices of G with $att_{G'}(e)$, the vertices previously incident to the a-edge.*

Theorem 4.4. *Recognizable graph languages are not closed under substitution.*

Proof. Again we will show this by providing a counter-example. First, let L be the language of all graphs that have exactly one vertex v and an arbitrary number of a-edges of arity 1. As a substitution graph we will use the graph G that consists of one vertex (that also is the merge-vertex), one b-edge and one c-edge, both of arity 1. This substitution rule applied to L generates the language of all graphs with one vertex, any number of b-edges and the same amount of c-edges. As before we see that this language is not recognizable, whereas L ist recognizable. □

5 Conclusion

We gave an explicit construction for an automaton functor which recognizes the concatenation of the languages of two given automaton functors. More precisely, given an automaton functor $\mathcal{A}_{X,K}$ accepting a language $L_{X,K}$ of cospans with domain X and codomain K and an automaton functor $\mathcal{A}_{K,Y}$ accepting a language $L_{K,X}$ of cospans with domain K and codomain Y, we constructed an automaton functor $\mathcal{A}$ which accepts the concatenation of $L_{X,K}$ and $L_{K,Y}$. The construction is non-trivial because we have to make sure that the constructed automaton functor is a functor, that is, if a cospan c can be decomposed into $c = c_1 ; c_2$, it must hold that $\mathcal{A}(c) = \mathcal{A}(c_1) ; \mathcal{A}(c_2)$. By giving this construction, we have shown that recognizable languages (of arrows in an adhesive category) are closed under concatenation.

One application of the construction could be the calculation of strongest post-conditions. Let a property be given as an automaton functor $\mathcal{A}$, and let $p = (\ell, r)$ be a graph rewriting rule in the format of reactive systems [LM00], that is, r and ℓ are cospans with the same domain and codomain, and the reduction relation is generated by $\ell ; c \Rightarrow r ; c$ for all context cospans c. We construct an automaton $\mathcal{A}'$ which accepts c if and only if $\mathcal{A}$ accepts $\ell ; c$ by changing the initial states of $\mathcal{A}$ to the states which are reachable by ℓ from an original initial state (see also [BBEK12]). Then we have two recognizable languages, $\{r\}$ and $L(\mathcal{A}')$ and we obtain the strongest post-condition by constructing the composition automaton. It is future research to make this idea more concrete. Note that in this application one of the recognizable languages is quite simple, consisting of only a single cospan. Another future plan is to find a more efficient construction for such simple concatenations.

In the final part of the paper we showed, by giving counter-examples, that recognizable languages are not closed under (a suitable notion of) Kleene star and substitution. Especially the second negative result is unfortunate, as a similar property for regular word languages played a key role in the match-bound technique for showing that a string rewrite system terminates [GHW03]. This makes it hard to transfer the technique to graph transformation systems.

Additional closure properties were considered in [Küp12], which forms the basis of this paper. Especially [Küp12] contains additional negative results and the proof that recognizable graph languages are closed under application of inverse context-free rules. Furthermore the construction of the concatenation automaton, given in this paper, is broken down to atomic cospans, a necessary requirement for the concrete construction of such automata, which we plan to implement in the tool RAVEN [BBEK12].

Bibliography

[BBC+11] Paolo Baldan, Filippo Bonchi, Andrea Corradini, Tobias Heindel, and Barbara König. A lattice-theoretical perspective on adhesive categories. *Journal of Symbolic Computation*, 46:222–245, 2011.

[BBEK12] Christoph Blume, H.J. Sander Bruggink, Dominik Engelke, and Barbara König. Efficient symbolic implementation of graph automata with applications to invariant checking. In *Proc. of ICGT '12 (International Conference on Graph Transformation)*, pages 264–278. Springer, 2012. LNCS 7562.

[BK08] H.J. Sander Bruggink and Barbara König. On the recognizability of arrow and graph languages. In *Proc. of ICGT '08 (International Conference on Graph Transformation)*, pages 336–350. Springer, 2008. LNCS 5214.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs I. Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[Cou94] Bruno Courcelle. Recognizable sets of graphs: Equivalent definitions and closure properties. *Mathematical Structures in Computer Science*, 4(1):1–32, 1994.

[GHW03] Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-bounded string rewriting systems. In Branislav Rovan and Peter Vojtas, editors, *Mathematical Foundations of Computer Science 2003*, volume 2747 of *Lecture Notes in Computer Science*, pages 449–459. Springer Berlin / Heidelberg, 2003.

[Hab92] Annegret Habel. Hyperedge Replacement: Grammars and Languages. *Lecture Notes in Computer Science*, 643, 1992.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Prentice Hall, 2006.

[Küp12] Sebastian Küpper. Abschlusseigenschaften für Graphsprachen mit Anwendungen auf Terminierungsanalyse. Master's thesis, Universität Duisburg-Essen, August 2012.

[LM00] James J. Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In *Proc. of CONCUR 2000*, pages 243–258. Springer, 2000. LNCS 1877.

[LS05] Stephen Lack and Paweł Sobociński. Adhesive and quasiadhesive categories. *ITA*, 39(3):511–545, 2005.