Electronic Communications of the EASST
Volume 29 (2010)

Proceedings of the Ninth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2010)

Recognizable Graph Languages for Checking Invariants

Christoph Blume, H.J. Sander Bruggink and Barbara König

13 pages

Guest Editors: Jochen Küster, Emilio Tuosto
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Recognizable Graph Languages for Checking Invariants

Christoph Blume, H.J. Sander Bruggink and Barbara König

Abteilung für Informatik und Angewandte Kognitionswissenschaft,
Universität Duisburg-Essen, Germany
christoph.blume@uni-due.de, sander.bruggink@uni-due.de, barbara_koenig@uni-due.de

Abstract: We generalize the order-theoretic variant of the Myhill-Nerode theorem to graph languages, and characterize the recognizable graph languages as the class of languages for which the Myhill-Nerode quasi order is a well quasi order. In the second part of the paper we restrict our attention to graphs of bounded interface size, and use Myhill-Nerode quasi orders to verify that, for such bounded graphs, a recognizable graph property is an invariant of a graph transformation system. A recognizable graph property is a recognizable graph language, given as an automaton functor. Finally, we present an algorithm to approximate the Myhill-Nerode ordering.

Keywords: graph transformation, recognizable graph languages, Myhill-Nerode theorem, invariants

1 Introduction

Regular languages and well quasi orders have proven to be useful analysis techniques in the field of string rewrite systems. In particular, the Myhill-Nerode well quasi order of a regular language L, which is strongly related to the well-known Myhill-Nerode equivalence, has nice properties [EHR83, LV94]: the left and right concatenation are monotone w.r.t. the order and the regular language L used to define it is upward-closed with respect to it. Let a string rewrite system $\mathscr{S}$ be given. From the first property it follows that if r is greater (with respect to the order) than ℓ for every rewrite rule $\ell \to r$ of $\mathscr{S}$, then it holds that v is greater than w for each word v reachable from w. The second property means, that for each word v that is greater than w, it holds that $v \in L$ if $w \in L$. Together, these two properties ensure that it is decidable whether a property, described as a regular language containing exactly the words satisfying the property, is an invariant of a string rewrite system.

Since the late 1980s several notions of regular graph languages – in this context called *recognizable* graph languages – have been introduced [BC87, Cou90, BK06, BK08b], which all turned out to be equivalent. Recognizable graph languages have found many applications, especially in the field of complexity theory.

In the light of the above observations it is natural to ask how results from regular languages, such as Myhill-Nerode equivalences, can be transferred and used for recognizable graph languages. While Myhill-Nerode equivalences are typically used to show that a language is not regular, we use them in a different way and study Myhill-Nerode quasi orders in order to verify that a specified property is an invariant of a graph transformation system.

The definition of recognizable graph language we use in this paper is based on the notion of automaton functor introduced in [BK08b], a category-based generalization of finite (word)

automata. Like finite automata in the word case, automaton functors provide an operational view on recognizable graph languages, which allows one to define a "Myhill-Nerode"-order on automaton states rather than on graphs directly. This is convenient, because states typically represent an infinite class of graphs. Still, automaton functors are in general infinite structures, due to the unboundedness of graph interfaces. In Section 2 we briefly define recognizable graph languages, automaton functors, and the category-theoretic notions at the heart thereof.

In Section 3 we generalize the order-theoretic variant of the Myhill-Nerode theorem to (recognizable) graph languages; that is, we define the Myhill-Nerode quasi order on graph languages and characterize recognizable graph languages as the class of languages for which this order is a well quasi order.

In the second part of the paper we focus on the application of the Myhill-Nerode quasi order in practice. First, in Section 4 we show that we need only define the automaton functor for a restricted set of so-called *atomic cospans*, so that we do not need consider *all* cospans when calculating the order.

As indicated above, the quasi order typically cannot be represented in a finite way, due to the unboundedness of graph interfaces. In Section 5 therefore, we restrict our attention to graphs which can be constructed with atomic cospans of bounded interface sizes, and we present an algorithm which approximates (and in the case of deterministic automaton functors even computes) the Myhill-Nerode quasi order of an automaton functor. Finally, we illustrate the work with a short example in Section 6. The full version with proofs can be found at [BBK10].

2 Preliminaries

In this section we briefly recall some concepts of category theory and recognizable graph languages. We presuppose a basic knowledge of category theory and order theory.

2.1 Category Theory and Recognizable Graph Languages

First we review and fix some notations. The category which has sets as objects, relations as arrows and relation composition as composition operator is denoted by $\mathcal{Rel}$. The subcategory which has total functions as arrows instead of relations is denoted by $\mathcal{Set}$. The composition of two arrows f and g will be denoted by ; where $f;g = g \circ f$ indicates the arrow which is obtained by first applying the arrow f and then the arrow g.

Let $\mathscr{C}$ be a category with pushouts. A cospan $c\colon J \xrightarrow{c^L} C \xleftarrow{c^R} K$ is a pair of $\mathscr{C}$-arrows with the same codomain. Here, J and K are the domain (or *inner interface*) and codomain (or *outer interface*) of the cospan c, respectively. The identity cospan for an object E is the cospan consisting of twice the identity arrow of E. Let $c\colon J \xrightarrow{c^L} C \xleftarrow{c^R} K$ and $d\colon K \xrightarrow{d^L} D \xleftarrow{d^R} M$ be cospans (where the codomain of c equals the domain of d). The composition of c and d is obtained by taking the pushout of c^R and d^L. A *semi-abstract cospan* is an equivalence class of cospans, where we take the middle object of the cospan up to isomorphism. Now, the cospan category $\mathit{Cospan}(\mathscr{C})$ is defined as the category which has the objects of $\mathscr{C}$ as objects, and semi-abstract cospans as arrows. If the middle object is not important, a cospan $c\colon J \to C \leftarrow K$ (an arrow in the cospan category from J to K) will be denoted as $c\colon J \nrightarrow K$.

Let a set Σ of labels be given. A *hypergraph* G, later also simply called *graph*, is a four-tuple $\langle V_G, E_G, \mathsf{att}_G, \mathsf{lab}_G \rangle$, where V_G is a finite set of *vertices* (or *nodes*) of G, E_G is a finite set of *edges* of G, $\mathsf{att}_G \colon E_G \to V_G^*$ is the *attachment function* and $\mathsf{lab}_G \colon E_G \to \Sigma$ is the *labeling function*. Here, V_G^* denotes the set of finite sequences of elements of V_G. A *hypergraph morphism* f is a structure-preserving map between two hypergraphs. A discrete graph is a graph which does not contain any edges. The discrete graph with n nodes is denoted by D_n. The *empty graph* is denoted by $\emptyset$ instead of D_0. The category of graphs and graph morphisms is denoted by $\mathcal{HG}raph$.

A cospan of graphs (an arrow in the category $Cospan(\mathcal{HG}raph)$) can be seen as a graph with an inner (left) and an outer (right) interface. Intuitively, the interfaces designate the parts of the graph which can be "touched" from the outside. With $[G] \colon \emptyset \to G \leftarrow \emptyset$ we denote the cospan consisting of a graph G with empty inner and outer interfaces.

Cospans of graphs are closely related to graph transformation systems, in particular to the double-pushout (DPO) approach to graph rewriting [SS05]. A DPO rewrite rule $\rho \colon L \leftarrow \rho_{\mathsf{L}} - I - \rho_{\mathsf{R}} \rightarrow R$ can be considered as a pair of cospans $\ell \colon \emptyset \to L \leftarrow \rho_{\mathsf{L}} - I$ and $r \colon \emptyset \to R \leftarrow \rho_{\mathsf{R}} - I$, which will in the following be called left- and right-hand side, respectively. Then it holds that $G \Rightarrow_\rho H$ if and only if $[G] = \ell ; c$ and $[H] = r ; c$, for some cospan c.

We define recognizable graph languages by using automaton functors on the category of cospans of graphs, as in [BK08b].

Definition 1 (Automaton functor, recognizability) Let a category $\mathscr{C}$ with initial object $\emptyset$ be given. An automaton functor is a functor $\mathscr{A} \colon \mathscr{C} \to \mathcal{R}el$, which maps every object X of $\mathscr{C}$ to a finite set $\mathscr{A}(X)$ of *states* of X and every arrow $f \colon X \to Y$ to a relation $\mathscr{A}(f) \subseteq \mathscr{A}(X) \times \mathscr{A}(Y)$, together with two distinguished sets $\mathrm{I}^{\mathscr{A}} \subseteq \mathscr{A}(\emptyset)$ and $\mathrm{F}^{\mathscr{A}} \subseteq \mathscr{A}(\emptyset)$ of *initial* and *final states*, respectively.

An automaton functor is *deterministic* if every relation $\mathscr{A}(f)$ is a function and every $\mathrm{I}^{\mathscr{A}}$ contains exactly one element.

An arrow $f \colon \emptyset \to \emptyset$ is accepted by an automaton functor $\mathscr{A}$, if $\langle s,t \rangle \in \mathscr{A}(f)$, for some $s \in \mathrm{I}^{\mathscr{A}}$ and $t \in \mathrm{F}^{\mathscr{A}}$. The language $L(\mathscr{A})$ of an automaton functor contains exactly those arrows which are accepted by it. A language L of arrows from $\emptyset$ to $\emptyset$ is a *recognizable language* if $L = L(\mathscr{A})$, for some automaton functor $\mathscr{A}$.

The intuition behind the definition is to have a mapping into a (locally) finite domain. The functor property guarantees that decomposing an object in different ways does not affect acceptance in any way. This is different from word languages, where there is essentially one way to decompose an object into subobjects.

Familiar constructions on finite automata, such as the determinization construction, can be easily generalized to automaton functors. Also, it was shown in [BK08b], that restricting to discrete interfaces does not affect the expressiveness of the formalism. Due to the latter result, we shall restrict to discrete interfaces in the rest of this paper.

The above definition can easily be generalized to accept languages between arbitrary objects. However, in our setting we require only languages from the initial object to the initial object.

A characterization of recognizable graph languages can be obtained in terms of recognizable languages in $Cospan(\mathcal{HG}raph)$:

Definition 2 (Recognizable graph language) A set L of graphs is a *recognizable graph language*,

if $[L] = \{[G] : \emptyset \to G \leftarrow \emptyset \mid G \in L\}$ is a recognizable language in $Cospan(\mathcal{HGraph})$.

In the following we will not distinguish between L, a language of graphs, and $[L]$, a language of (cospans of) graphs with empty interfaces.

2.2 Orders on categories

One of the basic concepts in checking invariants of regular languages is the notion of (well) quasi orders. First, we review the definition of (well) quasi orders on arbitrary sets (see also [LV94]).

A *quasi order (qo)* is a binary relation $\sqsubseteq_M$ on a set M if $\sqsubseteq_M$ is *reflexive* and *transitive*. A quasi order $\sqsubseteq_M$ on M is called *well-quasi order (wqo)* whenever if $m_1, m_2, \ldots$ is an infinite sequence of elements of M, then there exist integers i, j such that $0 < i < j$ and $m_i \sqsubseteq m_j$. In the following we will write $\sqsubseteq$ instead of $\sqsubseteq_M$ if M is clear from the context.

Next, we consider a semigroup $(M, *)$ and a quasi order $\sqsubseteq$ on M. We say that $\sqsubseteq$ is *left-monotone* (resp. *right-monotone*) if for all $m_1, m_2, m \in M$ the following condition is satisfied:

$$m_1 \sqsubseteq m_2 \implies m * m_1 \sqsubseteq m * m_2 \quad (\text{resp.} \quad m_1 \sqsubseteq m_2 \implies m_1 * m \sqsubseteq m_2 * m).$$

In the following we will define orders on the homsets of a category. More specifically, two arrows f, g can only be related by a quasi order $\sqsubseteq$ if they have the same source and target objects. Alternatively we could consider $\sqsubseteq$ as a family of quasi orders, one for each homset.

The notion of order in categories is also present in enriched categories [GMM94, Kel82]. Note however that unlike in enriched categories we do not necessarily require that the order is always preserved by composition ($f \sqsubseteq f'$ and $g \sqsubseteq g'$ implies $f;g \sqsubseteq f';g'$), since we will usually only require right-monotonicity as defined above.

3 A Generalization of the Myhill-Nerode Theorem

In this section we generalize the theorem of Myhill-Nerode to graph languages. This theorem says that a language is regular if and only if it is the union of equivalence classes of a monotone (or right-monotone) congruence on words of finite index. There is an order-theoretic variant of this theorem given in [EHR83, LV94] saying that a language is regular if and only if it is upward-closed with respect to a monotone well quasi order.

In order to state this theorem in our framework we first need the notion of Myhill-Nerode quasi order. Note that while the word or string variant of this theorem uses orders that are both left-monotone and right-monotone, here we work only with right-monotone orders. Intuitively this is sufficient since we start with the empty interface and attaching any cospan on the left can always be simulated by attaching an appropriate cospan on the right.

Definition 3 (Myhill-Nerode quasi order) Let L be a graph language over $Cospan(\mathcal{HGraph})$. A quasi order $\leq_L$ on $Cospan(\mathcal{HGraph})$ is called *Myhill-Nerode quasi order (relative to L)*, if for arbitrary cospans $a, b : \emptyset \nrightarrow D_n$ the following condition is satisfied:

$$a \leq_L b \quad \text{iff} \quad \forall (c : D_n \nrightarrow \emptyset) : \ ((a;c) \in L \implies (b;c) \in L).$$

Based on $\leq_L$ we can define the *Myhill-Nerode equivalence* $\equiv_L$ on cospans $a, b\colon \emptyset \nrightarrow D_n$ as follows:

$$a \equiv_L b \quad \text{iff} \quad a \leq_L b \text{ and } a \geq_L b$$

The Myhill-Nerode equivalence is called *locally finite*, if for every cospans $a\colon \emptyset \nrightarrow D_n$ the equivalence class of a is a finite set.

One can prove that the Myhill-Nerode quasi order is in fact a quasi order on $Cospan(\mathcal{HG}raph)$. It also possesses two other properties which will be important in the following. (Note that all proofs can be found in the appendix.)

Proposition 1 *Let L be a graph language over $Cospan(\mathcal{HG}raph)$. The Myhill-Nerode quasi order (relative to L) is right-monotone and the language L is upward-closed with respect to $\leq_L$.*

This proposition is the key to invariant checking. We say that a graph language L is an invariant for a rule ρ if $G \in L$ and $G \Rightarrow_\rho H$ always implies $H \in L$.

Imagine a rule ρ is given by a pair of cospans $\ell, r\colon \emptyset \nrightarrow I$ and it holds that $\ell \leq_L r$. If G is rewritten to H via ρ we have that $[G] = \ell; c$ and $[H] = r; c$ for some cospan $c\colon I \nrightarrow \emptyset$. Now $\ell \leq_L r$ implies $[G] \leq_L [H]$ (right-monotonicity) and if G is contained in L, then H is contained in L as well (upward-closure). Hence L is an invariant w.r.t. ρ. Furthermore if $\ell \not\leq_L r$, there is a cospan c violating the condition of Definition 3 and L is no invariant w.r.t. ρ. Hence we have that L is an invariant for ρ if and only if $\ell \leq_L r$.

Similar to the case of word languages we can characterize the recognizable graph languages in terms of congruence classes as shown in [BK08b]. Furthermore Ehrenfeucht et al. [EHR83] give a generalization of the Theorem of Myhill-Nerode by characterizing regular languages in terms of well quasi orders instead of equivalence classes of finite index. As an important result we can lift this theorem to the case of recognizable graph languages.

Theorem 1 (Generalized Myhill-Nerode Theorem) *Let a graph language L over $Cospan(\mathcal{HG}raph)$ be given. The following statements are equivalent:*

(i) *L is a recognizable graph language,*

(ii) *$\equiv_L$ is locally finite and L is the union of (finitely many) equivalence classes of $\equiv_L$.*

(iii) *L is upward closed with respect to some right-monotone well quasi order $\sqsubseteq_L$.*

(iv) *The Myhill-Nerode quasi order $\leq_L$ is a well quasi order.*

4 Atomic Cospans

In this section we introduce atomic graph operations which play the role of letters in the case of words. These atomic graph operations are based on the algebra of graphs originally described by Courcelle [BC87]. Each atomic graph operation is given by an atomic cospan, so that applying the graph operation to a cospan (a graph with interfaces) amounts to composing the cospan with

the atomic cospan of the operation. In the following, we will not distinguish between graph operations and atomic cospans used to define them.

We assume that the set of nodes of each discrete graph D_n is $V_{D_n} = \{v_0, \dots v_{n-1}\}$. We set $\mathbb{N}_n = \{0, \dots, n-1\}$ and we denote the *disjoint union of two graphs G_1 and G_2* by $G_1 \oplus G_2$. We assume that G_1 and G_2 are disjoint. Furthermore we define the *disjoint union $f \oplus g\colon G_1 \oplus G_2 \to H_1 \oplus H_2$ of two graph morphisms $f\colon G_1 \to H_1$ and $g\colon G_2 \to H_2$* where H_1 and H_2 are disjoint as follows:

$$(f \oplus g)(v) = \begin{cases} f(v), & \text{if } v \in V_{G_1} \\ g(v), & \text{if } v \in V_{G_2} \end{cases} \quad \text{and} \quad (f \oplus g)(e) = \begin{cases} f(e), & \text{if } e \in E_{G_1} \\ g(e), & \text{if } e \in E_{G_2} \end{cases}.$$

Definition 4 (Atomic graph operations) *Restriction of the outer interface:* Let $\rho\colon D_{n-1} \to D_n$ with $\rho(v_i) = v_i$ be an arrow between two discrete graphs. We define the cospan res_n as follows: $\mathsf{res}_n\colon D_n \xrightarrow{\mathsf{id}_{D_n}} D_n \xleftarrow{\rho} D_{n-1}$.

Permutation of the outer interface: Let a permutation $\pi\colon \mathbb{N}_n \to \mathbb{N}_n$ with $\pi(i) = i+1$ for $0 \leq i < n-1$ and $\pi(n-1) = 0$ and an arrow $\sigma\colon D_n \to D_n$ with $v_i \mapsto v_{\pi(i)}$ between two discrete graphs be given. We define the cospan perm_n as follows: $\mathsf{perm}_n\colon D_n \xrightarrow{\mathsf{id}_{D_n}} D_n \xleftarrow{\sigma} D_n$.

Transposition of the outer interface: Let a transposition $\tau\colon \mathbb{N}_n \to \mathbb{N}_n$ with $\tau(0) = 1$, $\tau(1) = 0$ and $\tau(i) = i$ for $2 \leq i \leq n-1$ and an arrow $\sigma\colon V_n \to V_n$ with $v_i \mapsto v_{\tau(i)}$ between two discrete graphs be given. We define the cospan trans_n as follows: $\mathsf{trans}_n\colon D_n \xrightarrow{\mathsf{id}_{D_n}} D_n \xleftarrow{\sigma} D_n$.

Fusion of two nodes of the outer interface: Let $n > 1$ and an equivalence relation $\theta = \mathsf{id}_{V_n} \cup \{(v_0, v_1), (v_1, v_0)\}$, an arrow θ_{map} which maps every node of D_n to its θ-equivalence class, and an arrow $\varphi\colon D_{n-1} \to D$ with $v_i \mapsto [\![v_{i+1}]\!]_\theta$, where D is the discrete graph with node set $\{[\![v]\!]_\theta \mid v \in V_n\}$, be given. We define the cospan fuse_n as follows: $\mathsf{fuse}_n\colon D_n \xrightarrow{\theta_{map}} D \xleftarrow{\varphi} D_{n-1}$.

Connection of a single hyperedge: Let an edge label $A \in \Sigma$, $m \in \mathbb{N}$ with $0 \leq m \leq n$ and a hypergraph H which consists of a single hyperedge h with arity m and labeled with A be given. We define the cospan $\mathsf{connect}_n^{A,m}$ as follows: $\mathsf{connect}_n^{A,m}\colon D_n \xrightarrow{e} H \oplus D_{n-m} \xleftarrow{e} D_n$ with $e(v_i) = \mathsf{att}_i(h)$ for $0 \leq i < m$ and $e(v_i) = v_{i-m}$ otherwise.

Disjoint union with a single node: We define the cospan vertex_n as follows: $\mathsf{vertex}_n\colon D_n \xrightarrow{d^L} D_{n+1} \xleftarrow{\mathsf{id}_{D_{n+1}}} D_{n+1}$ with $d^L = \mathsf{id}_{D_n} \oplus i$ and $i\colon \emptyset \to D_1$.

The intuitions behind these atomic graph operations are as follows (see Figure 1): With the cospan res_n we can hide the last node of the outer interface of a precomposed cospan. The cospan fuse_n glues the first two nodes of the outer interface of a precomposed cospan and afterward restricts the second node of this outer interface.

The cospans trans_n and perm_n permute the outer interface of a precomposed cospan. The former maps the nodes of the outer interface in such a way that only the first two nodes are transposed. The latter permutes the nodes of the outer interface such that every node is mapped to its successor node.

In order to be able to construct new graphs the cospans vertex_n and $\mathsf{connect}_n^{A,m}$ can be used to generate new nodes and edges. By composing vertex_n with an arbitrary cospan $c\colon \emptyset \to G \leftarrow D_n$

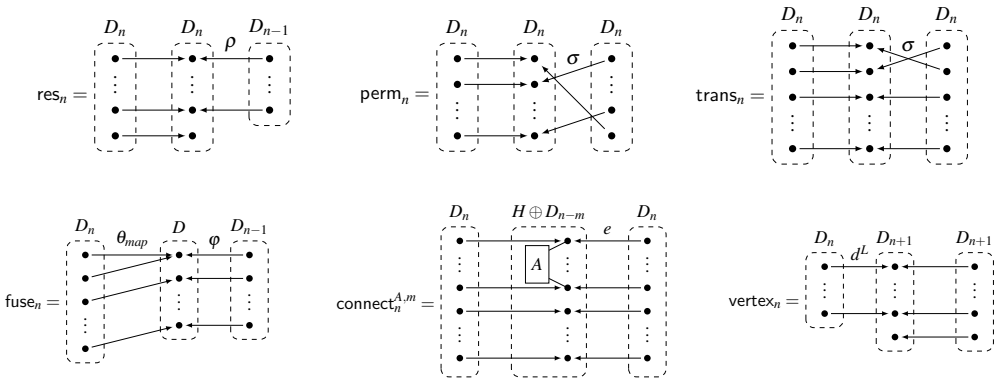


Figure 1: Graph operations

we add a single, isolated node to G and extend the outer interface of c to D_{n+1}, such that the last node of the extended outer interface is mapped to the new node. The cospan $\mathsf{connect}_n^{A,m}$ adds an A-labeled hyperedge with arity m in such a way to G that the first m nodes of the outer interface are mapped to the m nodes of the hyperedge h.

We can restrict our attention to these atomic graph operations, because any graph G (seen as a cospan of the form $\emptyset \to G \leftarrow \emptyset$) can be constructed by composing a finite number of them as shown by the next proposition.

Proposition 2 *Every cospan of the form* $c\colon D_m \to G \leftarrow D_n$ *where the right leg* φ^R *is injective can be constructed by a sequence* $\mathrm{op}_1,\ldots,\mathrm{op}_k$ *of atomic graph operations, i.e.* c *can be obtained as the composition* $c = \mathrm{op}_1;\ldots;\mathrm{op}_k$.

5 A Decidable Variant

In this section we develop an algorithm – based on the Myhill-Nerode quasi order – for checking invariants for recognizable graph languages. The algorithm takes as input an automaton functor which accepts the given graph language. In general this automaton functor has infinitely many states, since for every interface D_n ($n \in \mathbb{N}$) there exists a set of states. But for practical purposes we need an automaton functor which is finite, i.e. has only a finite number of states.

In order to get automaton functors with a finite number of state sets, we only take cospans with a bounded interface size into account.

Definition 5 (Bounded cospan) A cospan $c\colon S \nrightarrow T$ is called *bounded (by k)*, if there exist graph operations $\mathrm{op}_1,\ldots,\mathrm{op}_j$ such that $c = \mathrm{op}_1;\ldots;\mathrm{op}_j$ and for every graph operation $\mathrm{op}_i\colon D_{n_i} \nrightarrow D_{m_i}$ for $1 \le i \le j$ it holds that $n_i, m_i \le k$.

Definition 6 (Bounded Myhill-Nerode quasi order) Let a natural number $k \in \mathbb{N}$ and a graph language L over $Cospan(\mathcal{HG}raph)$ be given. The quasi order $\le_L^k$ on $Cospan(\mathcal{HG}raph)$ is called *bounded Myhill-Nerode quasi order (relative to L)*, if for arbitrary k-bounded cospans $a, b\colon \emptyset \nrightarrow D_n$

the following condition is satisfied:

$$a \leq_L^k b \quad \text{iff} \quad \forall (c\colon D_n \nrightarrow \emptyset,\ c\ k\text{-bounded})\colon\ ((a;c) \in L \implies (b;c) \in L)\,.$$

The bounded Myhill-Nerode quasi order defined above gives us an over-approximation of $\leq_L$, i.e., two cospans with $a \leq_L b$ are for sure related by the relation $\leq_L^k$, but not necessarily vice versa.

Note that graphs with edges of arity more than k can not be constructed by cospans that are bounded by k. Also for edges with smaller arity it is not guaranteed that they are constructible. For example a k-grid consisting of binary edges needs interfaces of size at least k.

Since all automaton functors which accept only cospans of bounded interface size have a finite representation, we are able to consider an algorithm which computes the Myhill-Nerode quasi order relative to a given deterministic automaton functor similar to the algorithm for computing the Myhill-Nerode equivalence by pairwise comparing two states with their successor states.

But for practical purposes the algorithm is not useful due to the fact, that in general the deterministic automaton functor can be exponentially larger than the equivalent non-deterministic automaton functor. Therefore we also allow non-deterministic automaton functors as input for the algorithm. However this leads to some additional changes. Since the automaton functor is non-deterministic, for a given state there exists a set of successor states instead of a unique successor state and we cannot pairwise compare two states with their (unique) successor states. In order to circumvent this difficulty, we allow an "one-sided error" by taking a stronger relation than the Myhill-Nerode quasi order. Roughly, we are under-approximating language inclusion via some form of simulation. A relation R on the states of an automaton functor $\mathscr{A}$ is a simulation, if the following condition is satisfied:

$$s_1\,R\,s_2 \implies \left(s_1 \in \mathrm{F}^{\mathscr{A}} \Rightarrow s_2 \in \mathrm{F}^{\mathscr{A}}\right) \wedge \forall \mathrm{op}\colon \forall s_1' \in \mathscr{A}(\mathrm{op})(s_1)\colon \exists s_2' \in \mathscr{A}(\mathrm{op})(s_2)\colon\ (s_1'\,R\,s_2').$$

A state t_2 *simulates* a state t_1, denoted by $t_1 \preceq t_2$, if $t_1\,R\,t_2$ holds for some simulation R.

Definition 7 (Bounded simulation) Let L be a graph language over $\mathit{Cospan}(\mathcal{H}\mathit{Graph})$ and $\mathscr{A}$ an automaton functor, which accepts the language L. The quasi order $\leq_{\mathscr{A}}^k$ is called *bounded simulation (relative to L)*, if for arbitrary, k-bounded cospans $a, b\colon \emptyset \nrightarrow D_m$ the following condition is satisfied:

$$a \leq_{\mathscr{A}}^k b \quad \text{iff} \quad \forall s_1 \in \mathscr{A}(a)(\mathrm{I}^{\mathscr{A}})\colon \exists s_2 \in \mathscr{A}(b)(\mathrm{I}^{\mathscr{A}})\colon\ s_1 \preceq s_2.$$

Replacing the (bounded) Myhill-Nerode quasi order by the (bounded) simulation relation results in fact in an one-sided error, as the next proposition shows:

Proposition 3 *Let $n, k \in \mathbb{N}$ with $n \leq k$, $a, b\colon \emptyset \nrightarrow D_n$ be cospans and $\mathscr{A}$ be the automaton functor which accepts the language L. If $a \leq_{\mathscr{A}}^k b$ holds, then $a \leq_L^k b$ holds. The inverse direction holds if $\mathscr{A}$ is deterministic.*

Algorithm 1 on page 9 computes $\leq_{\mathscr{A}}^k$ as defined above. Note that this is a fixed-point algorithm computing the greatest fixed-point. The relations $\preceq^i$ (one for each interface size) first contain all possible pairs of states and are suitably refined in each step. First, we delete all pairs, where the first state is final and the second is not. Then, for all pairs still in the relation we check whether

each transition from the first state can be mimicked by the second such that the resulting states are in the relation. If no more pairs can be deleted we have reached a fixed-point and terminate. Then it is left to check whether the two cospans under consideration are related.

Algorithm 1 `CheckSimRelated`$(a,b,k,\mathscr{A})$

Input: Bounded cospans $a,b\colon \emptyset \nrightarrow D_n$ with $n \leq k$, an automaton functor $\mathscr{A}$
Output: **true**, if $a \preceq^k_{\mathscr{A}} b$ and **false**, if $a \not\preceq^k_{\mathscr{A}} b$

```
set ⪯^i = A(D_i) × A(D_i) for all 0 ≤ i ≤ k
for all s_0 ∈ F^A, s_1 ∈ A(∅) \ F^A do
    delete (s_0, s_1) ∈ ⪯^0
repeat
    for all (s_0, s_1) ∈ ⪯^i with 0 ≤ i ≤ k do
        for all op ∈ {connect_i^{A,m}, fuse_i, perm_i res_i, trans_i, vertex_i} do
            for all s'_0 ∈ A(op)(s_0) do
                if there exists no s'_1 ∈ A'(op)(s_1), such that (s'_0, s'_1) ∈ ⪯^i then
                    delete (s_0, s_1) from ⪯^i
until no deletion has been performed in the last iteration
for all i ∈ I^A do
    for all s_0 ∈ A(a)(i) do
        if there exists no state s_1 ∈ A(b)(i), such that (s_0, s_1) ∈ ⪯^n then
            return false
return true
```

Theorem 2 *Let an automaton functor $\mathscr{A}$ and two bounded cospans $a,b\colon \emptyset \nrightarrow D_n$ with $n \leq k$ be given. Then $a \preceq^k_{\mathscr{A}} b$ holds, if and only if* `CheckSimRelated`$(a,b,k,\mathscr{A})$ *returns* `true`.

We implemented the algorithm in a naive way: our implementation explicitly stores the relations $\preceq^i$ in tables and iterates until no further changes occur. More details about the run-time and memory requirement of the naive implementation are given in the next section; some ideas for significant improvement are presented as future work in the conclusion.

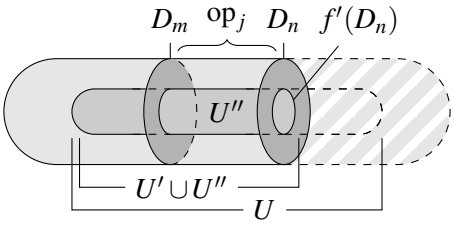
6 Short Example

In this section we consider a multi-user file system where the access to the system is controlled by several rules in order to guarantee some consistency properties. The case study was inspired by [KMP02]. As in most cases, the violation of these consistency properties can be modeled by the occurrence of one or more forbidden graphs. Therefore, we first introduce a k-bounded automaton functor $\mathscr{A}$, i.e. an automaton functor processing only k-bounded graphs, which accepts every graph $[G]$ which contains a specified subgraph U.

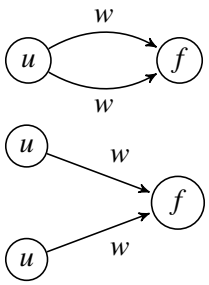
The idea behind this automaton functor is as follows: The automaton functor used in this example contains a state set $\mathscr{A}(D_i)$ for every discrete interface D_i, $0 \leq i \leq k$. Every state in each state set stores two kinds of information: on the one hand the subgraph U' of U which has already been read and on the other hand a partial function f from V_{D_i} to $V_{U'}$ describing which vertices of U' are contained in the interface D_i. By Proposition 2, we can restrict the automaton functor to

accept only atomic graph operations (see Section 4), since every cospan $[H]$ can be decomposed to a sequence of atomic graph operations $\mathrm{op}_1, \ldots, \mathrm{op}_\ell$ such that $[H] = \mathrm{op}_1 ; \ldots ; \mathrm{op}_\ell$. For every atomic graph operation $\mathrm{op}_j \colon D_m \nrightarrow D_n$ with $1 \leq j \leq \ell$, $m,n \in \{0,\ldots,k\}$ containing a subgraph U'' of U and a state $(U', f) \in \mathscr{A}(D_m)$ the successor state $(U' \cup U'', f') \in \mathscr{A}(D_n)$ is computed by adding the new subgraph U'' to the subgraph U' and updating the partial function f according to op_j resulting in the partial function f' (see image below). Note that op_j might contain various subgraphs U'' and hence the automaton is heavily non-deterministic. More details concerning the construction of this automaton functor can be found in [Blu08].

We can show that we obtain a functor which guarantees that the decomposition of the cospan $[H]$ does not affect the acceptance behavior of the automaton functor. The set of start states $\mathrm{I}^{\mathscr{A}}$ contains only the state $(\emptyset, \emptyset)$ consisting of the empty graph and the empty partial function. The set of acceptance states $\mathrm{F}^{\mathscr{A}}$ contains only the state $(U, \emptyset)$ consisting of the wanted subgraph and the empty partial function.

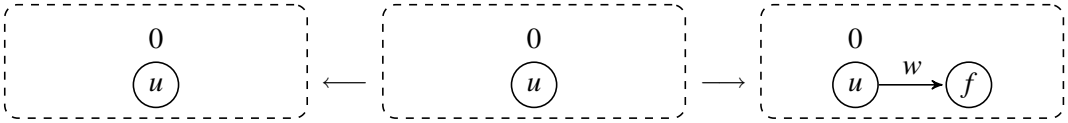


Now we want to use this automaton functor for the verification of the multi-user file system. We consider two properties which describe when the consistency of the multi-user file system is violated. The system is in a consistent state as long as these properties are *not* satisfied. The first property is the double write access of a user to a file (*double access*), i.e. a user has two times a write access to the same file at the same time. The second property is the write access of two different users to the same file at the same time (*two users*). These two properties can be modeled by the following two graphs, where nodes labeled with u (resp. f) denote users (resp. files) and edges from a user-node to a file-node labeled with w (resp. r) denote a write (resp. a read) access of that user to that file:

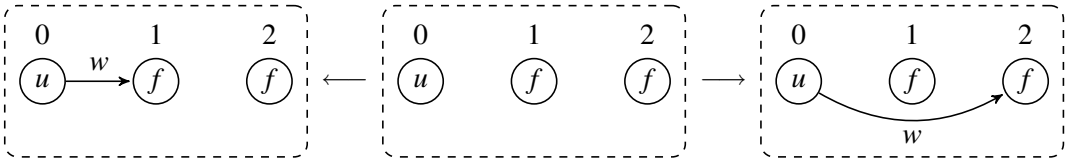


Note that it is not forbidden that a user has more than one read access to a file at the same time and that two or more users can have read access to the same file at the same time even if one user has write access to that file. Since recognizable languages are closed under boolean operations and with the considerations above we can now construct an automaton functor that recognizes all graphs violating one of the two properties, i.e., all graphs that contain either of the two subgraphs.

Furthermore, the multi-user file system offers the usual operations such as adding and removing users, creating, deleting and requesting files as well as switching, dispossessing and transferring access rights. In the following, we will show with the rules "User creates new file" and "User requests file" how these file system operations can be modeled as DPO rewrite rules. The rule "User creates new file" applied for some user u creates a new file f and gives the user a write access to this file. It can be modeled by the following span:



The rule "User requests file" applied for some user u sets the write access of this user from the current file to some other existing file. The following span models this rule:



Since every rewrite rule can be considered as two cospans ℓ and r (see Subsection 2.1) which are the left and right hand side of the corresponding rewrite rule, we can verify the consistency of this multi-user file system by checking, if the language of all graphs containing none of the forbidden subgraphs is an invariant for each rule. Since the automaton functor accepts the complement of this language, i.e., all graphs that *do* contain one of the forbidden subgraphs, we perform a backwards analysis on each rewrite rule and check whether $r \leq^k_{\mathscr{A}} \ell$. If r is related to ℓ, then the original rewrite rule does not violate the consistency of the multi-user file system. After the application of the rule the consistency of the system is violated only if it was already violated before the rule application, hence the language is verified to be an invariant.

We now use the algorithm described in the previous section to check the rewrite rules mentioned above. For all interface sizes that we checked the result of the algorithm is that the language is an invariant w.r.t. the first rule, but not w.r.t. the second rule. This is clear, since a user can request write access to a file, to which another user has already write access. Note also that, due to the under-approximation by simulations, there are actually rules which are correct, but are not recognized as such by the algorithm.

Although the example is rather small, the computed simulation relation becomes very large quickly. Table 1 presents the size of the simulation relation (according to the number of pairs contained in the relation) and the run-time of the implementation of Algorithm 1 for some interface sizes. The tests were performed on a Linux machine with a Xeon Dualcore 5150 processor and 2 GB of available main memory.

	Maximum interface size				
	0	1	2	3	4
Size (in pairs)	400	3.425	31.314	323.995	$\approx 3,7 \cdot 10^6$
Run-time (in seconds)	<1s	<1s	<1s	2s	26s

Table 1: Size of the simulation relation and run-time of the algorithm

Note that for interfaces with a size more than 4 the size of the simulation relation exceeds the amount of main memory. Nevertheless it is possible to verify all rewrite rules which have a interface size up to 4.

7 Conclusions

The notion of recognizable graph language used in this paper has been introduced in [BK08b] and is strongly related to [Cou90, Gri03, BK06]. Especially the notion of recognizability considered

here is equivalent to Courcelle's notion. For a detailed comparison see [BK08b]. In [BK08a] a weaker notion of graph automata is introduced.

Invariant checking for graph transformation rules has already been considered in several papers: in [FL97, BPR03] shape types and shapes are introduced in order to describe graph languages. Both papers propose algorithms that analyze each rule and check whether (and how) it may change the shape of a graph. In order to describe shapes the former uses context-free grammars whereas the latter uses more expressive graph reduction systems, that are able to express properties such as balancedness of trees. In [HPR06] a method for computing weakest preconditions of application conditions, which are equivalent to first-order graph logic, is presented. This method can also be used for invariant checking, by showing that for every rule the weakest precondition of the invariant is implied by the invariant. Note that, in general, recognizable graph languages are more expressive than first-order logic since every monadic second-order graph logic formula is known to specify a recognizable graph language [Cou90]. Another related work [BBG+06] considers graph patterns consisting of negative and positive components and shows that they are invariants via an exhaustive search. Interestingly, this method made efficient by a symbolic algorithm based on binary decision diagrams, an idea that we are trying to reuse in a somewhat different setting (see remarks below).

We have not yet compared the effectiveness of our approach to these other approaches in detail, but our method is different from all the others in that it is based on the Myhill-Nerode quasi order.

Our approach suffers from the restriction that we have to work with k-bounded cospans. Especially we first over-approximate the relation $\leq_L$ by $\leq_L^k$ (by introducing k-boundedness), which is subsequently under-approximated by $\leq_{\mathscr{A}}^k$ (by using simulation instead of language inclusion). While it is difficult to imagine how to avoid the restriction to interfaces up to size k, the determinization of the automaton functor $\mathscr{A}$, which would avoid the under-approximation, should be achievable if we use a more succinct representation of automaton functors. We are currently experimenting with the representation of automaton functors (which are basically very large relations) with binary decision diagrams (BDDs), which are well-suited for the compact representation of large (but finite) relations. Our experiments have so far been very promising. With BDDs we can handle much larger interfaces and we expect to obtain less memory usage and better run-times.

Finally, decomposing a graph into atomic cospans is basically equivalent to the path decomposition of a graph and checking whether a graph is contained in the language is hence linear-time for graphs of bounded pathwidth. For efficiency reasons it would be more suitable to consider generalizations of tree automata that can handle tree decompositions of graphs, as it is similarly done in the work by Courcelle. Hence we are currently investigating tree automata and their generalization to graphs.

Bibliography

[BBG+06] B. Becker, D. Beyer, H. Giese, F. Klein, D. Schilling. Symbolic invariant verification for systems with dynamic structural adaptation. In *Proc. of ICSE '06 (International Conference on Software Engineering)*. Pp. 72–81. ACM, 2006.

[BBK10] C. Blume, H. S. Bruggink, B. König. Recognizable Graph Languages for Checking

Invariants. 2010.
http://www.ti.inf.uni-due.de/publications/blume/invcheck.pdf

[BC87] M. Bauderon, B. Courcelle. Graph Expressions and Graph Rewritings. *Mathematical Systems Theory* 20(2-3):83–127, 1987.

[BK06] S. Bozapalidis, A. Kalampakas. Recognizability of graph and pattern languages. *Acta Informatica* 42(8/9):553–581, 2006.

[BK08a] S. Bozapalidis, A. Kalampakas. Graph automata. *Theoretical Computer Science* 393:147–165, 2008.

[BK08b] H. J. S. Bruggink, B. König. On the Recognizability of Arrow and Graph Languages. In *Proc. of ICGT '08*. Springer, 2008. LNCS 5214.

[Blu08] C. Blume. Graphsprachen für die Spezifikation von Invarianten bei verteilten und dynamischen Systemen. Master's thesis, Universität Duisburg-Essen, November 2008. (in German).

[BPR03] A. Bakewell, D. Plump, C. Runciman. Checking the Shape Safety of Pointer Manipulations. In *Proc. of RelMiCS '03*. Pp. 48–61. 2003.

[Cou90] B. Courcelle. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Inf. Comput.* 85(1):12–75, 1990.

[EHR83] A. Ehrenfeucht, D. Haussler, G. Rozenberg. On Regularity of Context-Free Languages. *Theor. Comput. Sci.* 27:311–332, 1983.

[FL97] P. Fradet, D. Le Métayer. Shape Types. In *Proc. of POPL '97*. Pp. 27–39. ACM, 1997.

[GMM94] P. H. B. Gardiner, C. E. Martin, O. de Moor. An algebraic construction of predicate transformers. *Sci. Comput. Program.* 22(1-2):21–44, 1994.

[Gri03] G. Griffing. Composition-representative subsets. *Theory and Applications of Categories* 11(19):420–437, 2003.

[HPR06] A. Habel, K.-H. Pennemann, A. Rensink. Weakest Preconditions for High-Level Programs. In *Proc. of ICGT '06*. Springer, 2006. LNCS 4178.

[Kel82] G. M. Kelly. *Basic Concepts of Enriched Category Theory*. Cambridge University Press, 1982.

[KMP02] M. Koch, L. V. Mancini, F. Parisi-Presicce. Decidability of Safety in Graph-based Models for Access Control. In *Proceedings of the 7th European Symposium on Research in Computer Security*. Pp. 229–243. Springer, 2002. LNCS 2502.

[LV94] A. de Luca, S. Varricchio. Well Quasi-Orders and Regular Languages. *Acta Inf.* 31(6):539–557, 1994.

[SS05] V. Sassone, P. Sobociński. Reactive Systems over Cospans. In *LICS*. Pp. 311–320. 2005.