Electronic Communications of the EASST
Volume 41 (2011)

Proceedings of the
Tenth International Workshop on
Graph Transformation and
Visual Modeling Techniques
(GTVMT 2011)

Decidability and Expressiveness of Finitely Representable Recognizable Graph Languages

H.J. Sander Bruggink and Mathias Hülsbusch

12 pages

Guest Editors: Fabio Gadducci, Leonardo Mariani
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Decidability and Expressiveness of Finitely Representable Recognizable Graph Languages

H.J. Sander Bruggink and Mathias Hülsbusch*

Universität Duisburg-Essen, Germany
sander.bruggink@uni-due.de, mathias.huelsbusch@uni-due.de

Abstract: Recognizable graph languages are a generalization of regular (word) languages to graphs (as well as arbitrary categories). Recently automaton functors were proposed as acceptors of recognizable graph languages. They promise to be a useful tool for the verification of dynamic systems, for example for invariant checking. Since automaton functors may contain an infinite number of finite state sets, one must restrict to finitely representable ones for implementation reasons. In this paper we take into account two such finite representations: *primitive recursive automaton functors* – in which the automaton functor can be constructed on-the-fly by a primitive recursive function –, and *bounded automaton functors* – in which the interface size of the graphs (cf. path width) is bounded, so that the automaton functor can be explicitly represented. We show that the language classes of both kinds of automaton functor are closed under boolean operations, and compare the expressiveness of the two paradigms with hyperedge replacement grammars. In addition we show that the emptiness and equivalence problem are decidable for bounded automaton functors, but undecidable for primitive recursive automaton functors.

Keywords: recognizable graph languages, automaton functors, verification, graph transformation

1 Introduction

In [BK08] automaton functors were introduced as an automaton model to accept recognizable graph languages, and it was shown that this notion is equivalent to Courcelle's [Cou90]. Automaton functors can be seen as a generalization of finite (word) automata to graphs (and other categories): graphs are decomposed into a sequence of *cospans* (which, intuitively, can be seen as graphs with interfaces or external nodes), which are then input into an automaton to decide whether they are accepted. Such an automaton model for graph languages is useful in practice to verify properties of graph transformation systems; in [BBK10] for example, automaton functors were used for invariant checking.

However, in general, automaton functors are infinite structures. For implementation purposes, we need to impose restrictions to automaton functors in order to ensure that they have finite representations. In this paper, we will explore two possible such restrictions: *primitive recursive automaton functors* (Section 3) and *bounded automaton functors* (Section 4).

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

In the case of primitive recursive automaton functors, we require that the automaton functor can be generated "on-the-fly" by a primitive recursive function. Although other computability classes would be possible (and interesting), the requirement that the function is primitive recursive ensures that the function is computable and total.

In the case of bounded recursive automaton functors, we bound the size of the interfaces of the cospans in the decomposition (which amounts to bounding the number of external nodes). Thereby we bound the pathwidth of the graphs which can be generated. If we bound the interface size in this way, we can specify finitely many cospans from which all (bounded) cospans can be composed. Thus, every bounded automaton functor can be explicitly represented.

Of these two restricted forms of automaton functor, we are particularly interested in: a) the expressive power of the formalisms in relation to each other, and in relation to hyperedge replacement grammars; and b) what decision problems of the restricted automaton functors are decidable, and how automaton functors can be constructively combined into other automaton functors.

2 Preliminaries

In this section we briefly recall some concepts of category theory, recognizable graph languages and computability theory, mainly in order to fix terminology and notation.

2.1 Categories and recognizable arrow languages

We presuppose a basic knowledge of category theory. For arrows $f\colon A \to B$ and $g\colon B \to C$, the composition of f and g is denoted $(f;g)\colon A \to C$. The category **Rel** has sets as objects and relations as arrows. Its subcategory **Set** has only the functional relations (functions) as arrows.

Let $\mathbf{C}$ be a category in which all pushouts exist. A *cospan in* $\mathbf{C}$ is a pair $\langle c_L, c_R\rangle$ of $\mathbf{C}$-arrows $J \xrightarrow{c_L} G \xleftarrow{c_R} K$. In such a cospan, J will be called the left or *inner interface*, while K will be called the right or *outer interface*. Composition of two cospans $\langle c_L, c_R\rangle, \langle d_L, d_R\rangle$ is computed by taking the pushout of the arrows c_R and d_L. Cospans are isomorphic if their middle objects are (such that the isomorphism commutes with the component morphisms of the cospan). Isomorphism classes of cospans are the arrows of so-called cospan categories. That is, for a category $\mathbf{C}$ with pushouts, the category $\mathbf{Csp}(\mathbf{C})$ has the same objects as $\mathbf{C}$. The isomorphism class of a cospan $J \xrightarrow{c_L} G \xleftarrow{c_R} K$ in $\mathbf{C}$ is an arrow from J to K in $\mathbf{Csp}(\mathbf{C})$.

In [BK08], an *automaton functor*, which is an acceptor for recognizable languages of arrows in a category, was defined as follows:

Definition 1 Let a category $\mathbf{C}$ with initial object $\varnothing$ be given. An automaton functor is a functor $\mathcal{A}\colon \mathbf{C} \to \mathbf{Rel}$, which maps every object $X \in \mathbf{C}$ to a finite set $\mathcal{A}(X)$ (the *state set of* X) and every arrow $f \in \mathbf{C}(X,Y)$ to a relation $\mathcal{A}(f) \subseteq \mathcal{A}(X) \times \mathcal{A}(Y)$, together with two distinguished sets $I_{\mathcal{A}} \subseteq \mathcal{A}(\varnothing)$ and $F_{\mathcal{A}} \subseteq \mathcal{A}(\varnothing)$ of *initial* and *final states*, respectively.

An arrow $f \in \mathbf{C}(\varnothing,\varnothing)$ is accepted by an automaton functor $\mathcal{A}$, if $\langle s,t\rangle \in \mathcal{A}(f)$, for some $s \in I_{\mathcal{A}}$ and $t \in F_{\mathcal{A}}$. The language $\mathrm{L}(\mathcal{A})$ of $\mathcal{A}$ contains exactly those arrows which are accepted by it. A language L of arrows from $\varnothing$ to $\varnothing$ is a *recognizable language* if $L = \mathrm{L}(\mathcal{A})$ for some automaton functor $\mathcal{A}$.

The rationale behind the notion of automaton functor is that objects (the arrows of the category) are decomposed in "smaller" objects, and then read sequentially by the automaton functor, like in the case of word automata. As such decompositions are in general not unique, the functor property is needed to make sure the result is independent of the decomposition.

Note, that we will later apply this definition to the category of graphs and restrict ourselves to *discrete* interfaces. See Subsection 2.3.

2.2 Graphs

A *hypergraph* over a set of labels Σ (in the following also simply called *graph*) is a structure $G = \langle V, E, att, lab \rangle$, where V is a finite set of nodes, E is a finite set of edges, $att\colon E \to V^*$ maps each edge to a finite sequence of nodes attached to it, and $lab\colon E \to \Sigma$ assigns a label to each edge. A *discrete graph* is a graph without edges; the discrete graph with k nodes (which is unique up to isomorphism) is denoted by D_k. A graph morphism is a structure preserving map between two graphs. The category of graphs and graph morphisms is denoted by **Graph**. Recall, that the *monomorphisms* (monos) and *epimorphisms* (epis) of the category **Graph** are the injective and surjective graph morphisms, respectively. Unless otherwise indicated, we will identify isomorphic graphs. Because of this, the collections of all graphs is a set rather than a proper class.

Graphically, nodes are represented by black circles. Edges are represented by a box with the label of the edge written in it. Open circles on the border of an edge denote "ports", where the edge must be connected to a node. For example:

A cospan $J \xrightarrow{c_L} G \xleftarrow{c_R} K$ in **Graph** can be viewed as a graph (G) with two interfaces (J and K), called the *inner interface* and *outer interface* respectively. Informally said, only elements of G which are in the image of one of the interfaces can be "touched", in the sense that they can be connected to or fused with other elements. By $[G]$ we denote the trivial cospan $\varnothing \to G \leftarrow \varnothing$, the graph G with two empty interfaces.

Cospans of graphs are intimately connected with the double pushout approach to graph rewriting [SS05]. A rewriting rule $p = \langle \ell, r \rangle$ can be defined as a pair of cospans $\ell\colon \varnothing \to L \leftarrow I$ and $r\colon \varnothing \to R \leftarrow I$ with the same outer interface. A graph G rewrites to H by applying rule p if and only if $[G] = \ell; c$ and $[H] = r; c$ for some cospan $c\colon I \to K \leftarrow \varnothing$ (where K is an arbitrary graph). This approach to graph transformation is easily seen to be equivalent to the double pushout approach.

Assume the label set Σ is partitioned into a set N of *non-terminals* and a set T of *terminals*. A handle of a non-terminal $A \in N$ is a cospan $h_A\colon \varnothing \xrightarrow{\ell} E_A \xleftarrow{r} J$, where J is a discrete graph, $E_A = \langle V, E, att, lab \rangle$ is a graph with $|V| = ar(A)$, $E = \{e\}$, $lab(e) = A$ and the elements of $att(e)$ are pairwise unequal, and the morphism r is injective and surjective on nodes. A *hyperedge replacement grammar* (HRG) is a set of rules in which the left-hand side is a handle, together with a start symbol $S \in N$. The language generated by an HRG $\mathcal{G}$ is defined as the reachable graphs containing only terminals: $\mathrm{L}(\mathcal{G}) = \{G = \langle V, E, att, lab \rangle \mid E_S \Rightarrow^* G \text{ for all } e \in E : lab(e) \in T\}$.[1] The class of languages expressible by hyperedge replacement grammars is denoted by HR.

[1] See [Hab92] for a different (but equivalent with respect to expressive power) definition of HRGs.

We define the category **CG** as the subcategory of **Csp**(**Graph**) which has discrete graphs as objects and (isomorphism classes of) cospans of graphs with discrete interfaces as arrows. Its subcategories $\mathbf{CG}_k$ (for $k \in \mathbb{N}$) have as objects only the discrete graphs with less than k nodes, and as arrows only the cospans which can be decomposed in atomic cospans with interfaces of size at most k.

2.3 Recognizable graph languages

When we apply Definition 1 to the category of cospans of graphs, we obtain an automaton model for recognizing graph languages. In [BK08], two important facts about this model where shown: first, that the model is equivalent to Courcelle's notion of recognizability [Cou90], in that the two paradigms recognize the same class of graph languages; and second, that restricting to discrete interfaces does not change the expressiveness of the model. In view of the latter, we will restrict to discrete interfaces in the rest of the paper (that is, we use the category **CG** in the definition below).

Definition 2 A set of graphs L is a *recognizable graph language* if there exists an automaton functor $\mathcal{A}\colon \mathbf{CG} \to \mathbf{Rel}$ such that $L = \{G \mid [G] \in \mathrm{L}(\mathcal{A})\}$. The class of recognizable graph languages is denoted by *Rec*.

Examples of recognizable graph languages are: graphs containing a graph G as a subgraph, connected graphs, k-colorable graphs (see Example 1), etc.

Example 1 (k-colorability) *Let $\mathbb{N}_k = \{0,\dots,k-1\}$ and G a graph. A k-coloring of G is a function $f\colon V_G \to \mathbb{N}_k$ such that for all $e \in E_G$ and for all $v_1, v_2 \in att_G(e)$ it holds that $f(v_1) \neq f(v_2)$ if $v_1 \neq v_2$. The following automaton functor $\mathcal{C}\colon \mathbf{CG} \to \mathbf{Rel}$ recognizes the k-colorable graphs:*

- *Every discrete graph J is mapped to $\mathcal{A}(J)$, the set of all valid k-colorings of J. Since J is discrete, this amounts to the entire function space from V_J to $\mathbb{N}_k$: $\mathcal{C}(J) = \mathbb{N}_k^{V_J}$.*
- *For a cospan $c\colon J \to G \leftarrow K$ the relation $\mathcal{C}(c)$ relates two colorings f_J, f_K, whenever there exists a coloring f for G such that $f(c_L(v)) = f_J(v)$ for every node $v \in V_J$ and $f(c_R(v)) = f_K(v)$ for every node $v \in V_K$.*

Specifically we have that $\mathcal{C}(\varnothing) = \{\varnothing\}$ where $\varnothing$ is the empty coloring. Then in order to accept all k-colorable graphs with empty interfaces we take $I_{\mathcal{C}}(\varnothing) = F_{\mathcal{C}}(\varnothing) = \{\varnothing\}$: a cospan $\varnothing \to G \leftarrow \varnothing$ is accepted whenever the two empty mappings are related.

The working of the automaton functor can be understood as follows. The automaton functor sequentially reads (a decomposition of) the graph. For each new node it encounters, it non-deterministically chooses a color. The graph is k-colorable if this is possible until the entire graph has been read.

In the context of the decidability results in this paper, we investigate the following decision problems concerning graph languages:

- *Word problem.* Given an automaton functor $\mathcal{A}$ and a graph G, decide whether $G \in \mathrm{L}(\mathcal{A})$. (In the present context, maybe this problem should be called *graph problem*, but we chose to stay consistent with the present formal languages literature.)
- *Emptiness problem.* Given an automaton functor $\mathcal{A}$, decide whether $\mathrm{L}(\mathcal{A}) = \varnothing$.

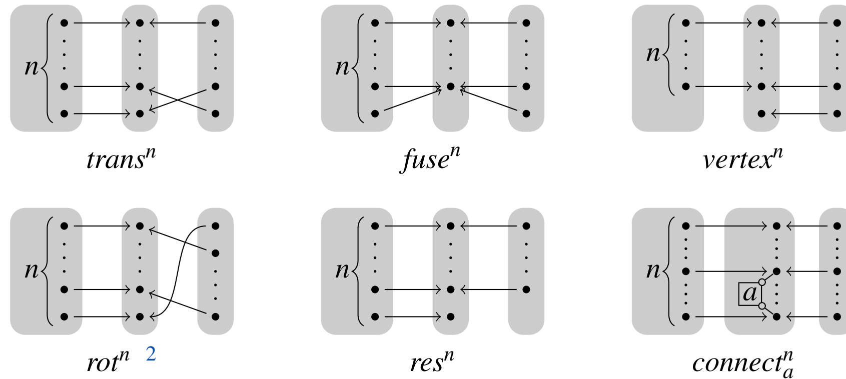


Figure 1: The atomic cospans.

– *Equivalence problem.* Given two automaton functors $\mathcal{A}$ and $\mathcal{B}$, decide whether $\mathrm{L}(\mathcal{A}) = \mathrm{L}(\mathcal{B})$.

Even between two fixed interfaces there exist infinitely many cospans. However, we can define a restricted set of cospans from which any graph can be composed, such that there is a finite amount of cospans for each pair of inner and outer interface. We call the cospans in this restricted set *atomic cospans*. There are different possible sets of atomic cospans; the specific version which is of use depends on the application. The atomic cospans we present here are similar to the atomic graphs of [GH97]. They have the advantage that not only every *graph*, but even every *cospan* can be composed from them. A slightly different set of atomic cospans, which ensures that the right morphism of the cospan is always injective, is presented in [BBK10].

The atomic cospans are presented in Figure 1. All atomic cospans are parametrized by the number of nodes in the inner (left) interface; the *connect* cospan is additionally parametrized by the label of the edge.

Cospans (and atomic cospans in particular) can be seen as operations on graphs. When we consider a graph with one interface J, modeled by a cospan $\varnothing \to G \leftarrow J$, post-composing with a cospan $J \to H \leftarrow K$ changes it into a graph with interface K by taking the disjoint union of G and H and fusing corresponding nodes. The atomic cospans correspond to the following actions: *trans* and *rot* change the positions of the node of the graph in the interface; *fuse* fuses two nodes to one; *res* removes one node from the interface (it stays in the graph); and *vertex* and *connect* add a node and an edge, respectively.

Definition 3

(i) An *atomic cospan decomposition* of a graph G is a sequence $\mathbf{c} = c_1, \ldots, c_n$ of atomic cospans, such that $cod(c_i) = dom(c_{i+1})$, for $1 \le i < n$, and $c_1 ; \cdots ; c_n = [G]$. If $\mathbf{c}$ is an atomic cospan composition of G, we will write $Graph(\mathbf{c}) = G$.

(ii) The interface size of an (atomic) cospan $c\colon D_j \to G \leftarrow D_k$ is defined as $|c|_{\mathrm{I}} = \max\{j,k\}$. The interface size of an atomic cospan decomposition $\mathbf{c}$ is defined as $|\mathbf{c}|_{\mathrm{I}} = \max\{|c|_{\mathrm{I}} \mid c \in \mathbf{c}\}$.

3 Primitive Recursive Recognizable Graph Languages

We first study primitive recursive recognizable graph languages. The idea is that the automaton functor which accepts the language is required to be primitive recursive, that is, given an arrow c, the relation it maps to can be calculated on-the-fly by a primitive recursive function. Thus, an automaton functor can be specified as a finite program in some suitable programming language.

Definition 4 A language L is *primitive recursive recognizable*, if $L = \mathrm{L}(\mathcal{A})$ for some primitive recursive automaton functor $\mathcal{A}\colon \mathbf{CG} \to \mathbf{Rel}$. The class of primitive recursive recognizable languages is denoted by *PRRec*.

3.1 Closure properties and decidability

We begin our discussion of primitive recursive graph languages by exploring their closure properties and decidability, which are both fundamental to their usefulness in practice.

The following two positive results (closure under boolean operations and decidability of the word problem) are easily proved.

Proposition 1 *The class of primitive recursive recognizable languages is closed under union, intersection and complement.*

Proof. It is easily seen, that the constructions used in the proof of Prop. 4.2 of [BK08] are all primitive recursive, and therefore the compositions are also primitive recursive. □

Proposition 2 *The word problem for primitive recursive recognizable languages is decidable, that is, given a primitive recursive automaton functor $\mathcal{A}$ and a graph cospan c, it is decidable whether or not c is accepted by $\mathcal{A}$.*

Proof. Since the automaton functor is primitive recursive, an thus computable, we can obtain $\mathcal{A}(c)$ and check whether it relates an initial with a final state. □

The main result of this section is that the emptiness, equivalence and finiteness problems for primitive recursive recognizable languages are *not* decidable, even in the category **CG** of cospans of graphs.

Theorem 1 *The emptiness problem for primitive recursive recognizable languages is not decidable, that is, given a primitive recursive automaton functor $\mathcal{A}$, it is undecidable whether $\mathrm{L}(\mathcal{A}) = \varnothing$.*

Proof. We reduce the undecidable problem of satisfiability in first-order logic (with equality) to the emptiness problem, thus showing undecidability of that.

A model of first-order logic is represented as a graph in the straight-forward way, that is, objects are represented by nodes and relations by (hyper)edges.

A formula φ of first-order logic is easily translated into first-order graph logic, or, which is

[2] The *rot* cospan is called *perm* in [BBK10].

more convenient in this case, into a formula φ' of (the first-order fragment of) the subobject logic of [BK10]. Using the construction in the before-mentioned paper, we can construct an equivalent automaton functor $\mathcal{A}_\varphi$. Since all constructions are primitive recursive, this automaton functor is primitive recursive. Now, φ is unsatisfiable (that is, $\neg\varphi$ is valid) if and only if $\mathrm{L}(\mathcal{A}_\varphi) = \varnothing$. □

Corollary 1 *The equivalence problem for primitive recursive recognizable graph languages (given as primitive recursive automaton functors) is undecidable.*

Proof. The language of a computable automaton functor is empty if and only if the automaton functor is equivalent to a computable automaton functor which accepts the empty language. The latter is easily given, so that we have reduced emptiness to equivalence. Thus, by Theorem 1, the equivalence problem for primitive recursive recognizable languages is undecidable. □

3.2 Expressiveness

In this subsection, we explore the expressiveness of primitive recurse automaton functors. In particular, we show that *PRRec* and *HR* are incomparable with respect to set inclusion. To this end, we first state and prove a pumping lemma for *PRRec*.

First, let a be an arbitrary atomic cospan other than *connect*. There exists a cospan a' such that $a\,;a' = id$. That is, every atomic cospan except *connect* can be undone. We will therefore measure an atomic cospan decomposition $\mathbf{c}$ as follows:

$$|\mathbf{c}|_A = \text{number of } connect_A\text{-cospans in } \mathbf{c}$$
$$|\mathbf{c}|_\mathrm{E} = \sum_{A\in\Sigma} |\mathbf{c}|_A$$

Lemma 1 *Let* $\mathbf{c}$ *be an atomic cospan decomposition, and* $G = Graph(\mathbf{c})$. *Let* $G = \langle V, E, att, lab\rangle$.

$$|\mathbf{c}|_A = |\{e \in E \mid lab(e) = A\}|$$
$$|\mathbf{c}|_\mathrm{E} = |E|.$$

Proof. By the observation that, since interfaces are discrete by definition, *connect*-cospans add a single edge to the graph and no other atomic cospan changes the number of edges in the graph. □

If we restrict an automaton functor[3] to atomic cospans of bounded interface size (see also Section 4, we obtain a structure which is basically a finite automaton. It is therefore not surprising that we can prove a similar pumping lemma. Compared to the usual pumping lemma for regular word languages, the pumping lemma below has an additional quantification (to deal with the restriction to a bounded interface size) and some notational clutter (to deal with the difference between graphs and atomic cospan decompositions).

Lemma 2 (Pumping Lemma) *Let L be a recognizable graph language. For all $k \in \mathbb{N}$ there exists a pumping constant $n \in \mathbb{N}$ such that all atomic cospan decompositions* $\mathbf{c}$ *with* $|\mathbf{c}|_\mathrm{I} \le k$, $|\mathbf{c}|_\mathrm{E} > n$ *and* $Graph(\mathbf{c}) \in L$ *can be written as* $\mathbf{c} = \mathbf{uvw}$ *such that*

[3] Note that this holds for *any* automaton functor, not only primitive recursive ones.

- $|\mathbf{uv}|_\mathrm{E} \leq n$,
- $|\mathbf{v}|_\mathrm{E} \geq 1$ *and*
- *for all i,* $\mathbf{v}^i$ *is well-defined and* $Graph(\mathbf{uv}^i\mathbf{w}) \in L$.

Proof. The proof proceeds analogously to the proof of the pumping lemma for regular word languages. Let an automaton functor $\mathcal{A}$ with $\mathrm{L}(\mathcal{A}) = L$ and a $k \in \mathbb{N}$ be given. We assume, without loss of generality, that the state sets $\mathcal{A}(\mathrm{D}_i)$, for $0 \leq i \leq k$, are mutually disjoint.

Let $T = \{\langle q_1, a, q_2\rangle \mid \langle q_1, q_2\rangle \in \mathcal{A}(a)$ for some atomic cospan $a\}$. For $t = \langle q_1, a, q_2\rangle \in T$, we define $src(t) = q_1$, $tgt(t) = q_2$, $ac(t) = a$. A *path* of $\mathcal{A}$ is a sequence $t_1, \ldots, t_m$, where, for $1 \leq i < m$, $tgt(t_i) = src(t_{i+1})$. Clearly, a graph G is accepted by $\mathcal{A}$ if and only if there exists a path $t_1, \ldots, t_m$ such that $src(t_1)$ is an initial state, $tgt(t_m)$ is a final state and $ac(t_1)\ldots ac(t_m)$ is an atomic cospan decomposition of G.

Let $U = \{t \in T \mid ac(t) = connect^i_a$ for some $i \leq k$ and $a \in \Sigma\}$. We choose the pumping lemma constant $n = |U|$.

Let $\mathbf{c} = c_1, \ldots, c_m$ be an atomic cospan decomposition with $|\mathbf{c}|_\mathrm{I} \leq k$, $|\mathbf{c}|_\mathrm{E} > n$ and $Graph(\mathbf{c}) \in L$. By the previous observation, there exists a path $\mathbf{t} = t_1, \ldots, t_m$ from an initial to a final state labeled with the atomic cospans of $\mathbf{c}$. By construction $\mathbf{t}$ contains $|\mathbf{c}|_\mathrm{E}$ elements of U. Since $|\mathbf{c}|_\mathrm{E} > |U|$, one some of those elements must occur in $\mathbf{t}$ twice. Let p and q be the smallest indices (with $p \neq q$) such that $t_p = t_q$. We take $\mathbf{u} = ac(t_1)\ldots ac(t_{p-1})$, $\mathbf{v} = ac(t_p)\ldots ac(t_{q-1})$ and $\mathbf{w} = ac(t_q)\ldots ac(t_m)$.

Since p and q are the smallest indices, $\mathbf{uv}$ does not contain duplicate *connect*s, so $|\mathbf{uv}|_\mathrm{E} \leq n$. Since $ac(t_p) = connect_a$, for some $a \in \Sigma$, $|\mathbf{v}|_\mathrm{E} \geq 1$. Finally, since $t_p = t_q$, it must hold that $tgt(t_{p-1}) = src(t_p) = src(t_q) = tgt(t_{q-1})$, and therefore $t_1 \ldots t_{p-1}(t_p \ldots t_{q-1})^i t_q \ldots t_m$ must also be a path from an initial to a final state, for all $i \in \mathbb{N}$. Thus $Graph(\mathbf{uv}^i\mathbf{w}) \in L$ for all $i \in \mathbb{N}$. □

Theorem 2 *The language classes PRRec and HR are not comparable with respect to set inclusion, that is:*
(i) PRRec $\not\subseteq$ *HR*
(ii) HR $\not\subseteq$ *PRRec*

Proof. (i) The class of all graphs is not generated by a HRG (see Theorems IV.3.3, IV.3.4 and IV.3.9 of [Hab92]). On the other hand, it is primitive recursive recognizable by the automaton functor which maps each cospan c to the complete relation of the respective state sets.

(ii) The (string-)graph language $L = \{a^n b^n c^n \mid n \geq 1\}$ is generated by an HRG (see Example I.3.6 of [Hab92]). As we will show by means of the pumping lemma, it is however not recognizable. Note that the path width of every graph in L is 2. Choose any $k \geq 2$, and let n be the constant given by the pumping lemma. We choose the following atomic cospan decomposition of $G = a^n b^n c^n$:

$$vertex^0, (vertex^1, connect^2_a, res^2)^n, (vertex^1, connect^2_b, res^2)^n, (vertex^1, connect^2_c, res^2)^n, res^1$$

Since the pumping will take place within the first n *connect* cospans, it follows together with Lemma 1 that for all $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ satisfying the conditions of the pumping lemma, $Graph(\mathbf{uv}^2\mathbf{w}) \notin L$. □

4 Bounded Recognizable Graph Languages

Bounded recognizable graph languages are accepted by bounded automaton functors, automaton functors for graph cospans which are only defined on interfaces of bounded size. By listing the relations for the atomic cospans (of which there are finitely many for each interface size) it is possible to explicitly represent this kind of automaton functor.

Definition 5 (Bounded automaton functor)

(i) A *k-bounded automaton functor* is an automaton functor $\mathcal{A}$, for the category $\mathbf{CG}_k$, such that there exists an automaton functor $\mathcal{A}'$ for $\mathbf{CG}$ such that $\mathcal{A}(X) = \mathcal{A}'(X)$, for all $X \in \mathbf{CG}_k$, and $\mathcal{A}(c) = \mathcal{A}'(c)$, for all $c \in \mathbf{CG}_k(X,Y)$, where $X,Y \in \mathbf{CG}_k$.

(ii) A language L of graph cospans is *k-bounded recognizable*, if there exists a k-bounded automaton functor $\mathcal{A}$ such that $L = \mathrm{L}(\mathcal{A})$. L is *bounded recognizable*, if it is k-bounded recognizable for some $k \in \mathbb{N}$.
The class of k-bounded recognizable languages is denoted by $BRec_k$ and the class of bounded recognizable graph languages by $BRec$.

Example 2 *We can restrict the automaton functor of Example 1 to cospans which are composable from atomic cospans of size m and less. Now the automaton functor can be represented explicitly by storing the transition relations for the atomic cospans only. The result will be that we only be able to recognize k-colorable graphs with pathwidth less than m.*

4.1 Closure properties and decidability

As in the last section, we start by studying the closure properties and decidability of bounded recognizable graph languages. Since a bounded automaton functor is basically a large finite automaton, all well-known constructions from formal (word) language theory still work.

Proposition 3 *If $L_1 \in BRec_k$ and $L_2 \in BRec_k$ then:*

(i) $L_1 \cup L_2 \in BRec_k$

(ii) $L_1 \cap L_2 \in BRec_k$

(iii) $\overline{L_1} \in BRec_k$

Proof. We can use the constructions of Prop. 4.2 of [BK08] to obtain a bounded automaton functor for the new language. □

Proposition 4 *All of the following decision problems are decidable for bounded recognizable graph languages (where a bounded recognizable graph language is given as a bounded automaton functor):*

(i) The word problem.

(ii) The emptiness problem.

(iii) The equivalence problem.

Proof. We assume that a bounded automaton functor is given by explicitly listing the transition relations for the atomic cospans. This means that an automaton functor is given as a finite

automaton labeled with atomic cospans. It is therefore not surprising, that the proofs ideas from automata theory can be used.

(i) Solving the word problem amounts to decomposing the input cospan to atomic cospans and looking up and composing the respective transition relations, all of which are computable.

(ii) The language of a finite automaton functor is empty if and only if there is no path from an initial state to a final state. Since the searching space is finite, this is decidable using Dijkstra's algorithm.

(iii) We can employ an algorithm similar to finite automata: construct two equivalent, deterministic minimal automaton functors and check if they are isomorphic. The minimization and determinization procedure of [BK08] also work for bounded automaton functors.

□

4.2 Expressiveness

We conclude the discussion on bounded recognizable graph languages by exploring their expressiveness. Not surprisingly, they form a proper subset of both the primitive recursive recognizable graph languages and the languages generated by HRGs.

Because we need this result in the proof of Theorem 3, which is in turn needed for an easy proof of strict inclusion of *BRec* in *Rec*, we first show the (not necessarily strict) inclusion.

Proposition 5 *$BRec \subseteq PRRec$.*

Proof. Follows from the fact that, by definition, $BRec = \bigcup_{k\in\mathbb{N}} BRec_k$ and the fact that a k-bounded automaton functor can be finitely represented by explicitly storing the transition relations for a finite set of atomic cospans (see [BBK10]). □

Theorem 3 *The class of bounded recognizable graph languages is properly contained in the class of graph languages generated by* HRGs*; that is $BRec \subset HR$.*

Proof. The fact that $BRec \neq HR$ directly follows from Proposition 5 and Theorem 2 (ii). It remains to show that $BRec \subseteq HR$.

Let $\mathcal{A}$ be an automaton functor bounded by k. We will construct a (linear) HRG $\mathcal{G}$ which generates the same language. Assume, without loss of generality, that the state sets $\mathcal{A}(\mathrm{D}_i)$ (for $i \leq k$) are mutually disjoint. We define the set of non-terminals N as $N = \bigcup_{i=0}^{k} \mathcal{A}(\mathrm{D}_i) \cup \{S\}$, where $S \notin \bigcup_{i=0}^{k} \mathcal{A}(\mathrm{D}_i)$.

The rules of the grammar $\mathcal{G}$ are presented in Figure 2. Now every execution path of $\mathcal{A}$ (starting from an initial state) corresponds to a derivation of $\mathcal{G}$ (starting from S) and vice versa. The only way to remove a non-terminal in the grammar, is to have a non-terminal labeled with a final state and apply the second rule. Thus, $\mathrm{L}(\mathcal{A}) = \mathrm{L}(\mathcal{G})$. □

Theorem 4 *The class of bounded recognizable languages is properly contained in the class of primitive recursive recognizable languages; that is $BRec \subset PRRec$.*

Proof. The fact that $BRec \subseteq PRRec$ was proven in Proposition 5. The fact that the inclusion is

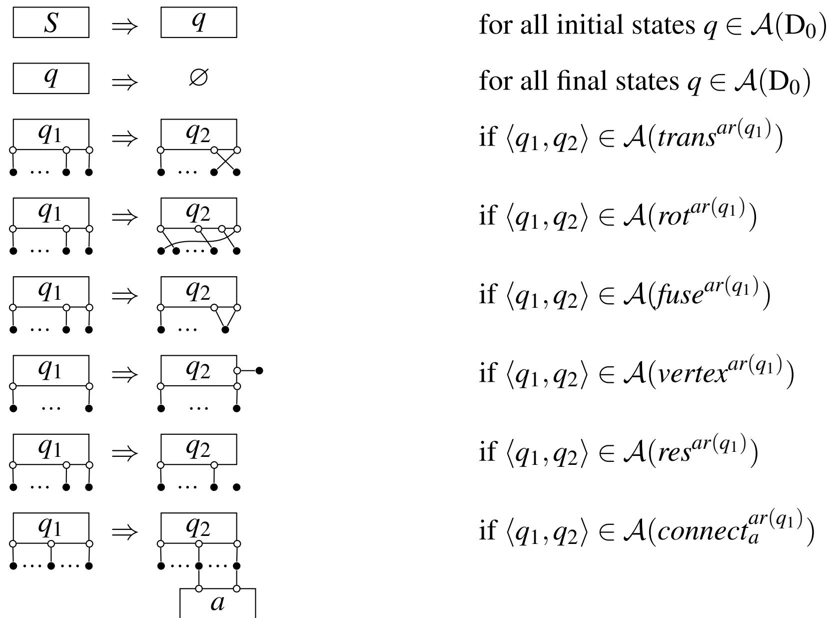


Figure 2: Rules of the $\mathcal{G}$ from the proof of Theorem 3.

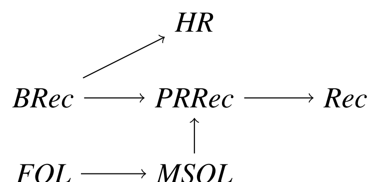
proper follows from Theorem 2 (i) and Theorem 3. □

5 Conclusion

In this paper we considered two restrictions of automaton functors for graphs that make them representable in a finite way, and as such implementable: *primitive recursive automaton functors*, which can be generated on-the-fly by a primitive recursive function, and *bounded automaton functors*, in which the interface size of the graphs is bounded.

We showed that the graph language classes accepted by both kinds of automaton functor are closed under the boolean operations and that the word problem of both kinds is decidable. However, of bounded automaton functors the emptiness and language equivalence problems are decidable, while these problems are undecidable for primitive recursive automaton functors. Another advantage of bounded automaton functors is that they can be *explicitly* represented, which is convenient when we want to apply certain algorithms on them.

How the expressiveness of the two paradigms is related to other common paradigms for the specification of graph languages, is summarized in the following diagram. Besides the language classes already mentioned in this paper, the diagram contains *FOL* and *MSOL*, the graph languages expressible by first-order logic and monadic second-order logic [Cou90] (for graphs equivalent with logic of subobjects [BK10]), respectively. An arrow from A to B means that every graph language expressible by A can also be expressed by B.



As is clear from the diagram, primitive recursive automaton functors are strictly more expressive than bounded automaton functors. Which of the representations is more convenient in practice depends on which of the above properties one needs for a given application.

Bibliography

[BBK10] C. Blume, S. Bruggink, B. König. Recognizable Graph Languages for Checking Invariants. In *Proc. of GT-VMT '10*. Electronic Communications of the EASST. 2010.

[BK08] S. Bruggink, B. König. On the Recognizability of Arrow and Graph Languages. In *Proc. of ICGT '08*. Pp. 336–350. Springer, 2008. LNCS 5214.

[BK10] S. Bruggink, B. König. A Logic on Subobjects and Recognizability. In *Proc. of IFIP-TCS '10*. Springer, 2010.

[Cou90] B. Courcelle. The Monadic Second-Order Logic of Graphs I. Recognizable Sets of Finite Graphs. *Information and Computation* 85:12–75, 1990.

[GH97] F. Gadducci, R. Heckel. An inductive view of graph transformation. In *Proceedings of WADT '97*. Pp. 223–237. 1997.

[Hab92] A. Habel. *Hyperedge Replacement: Grammars and Languages*. Springer, 1992.

[SS05] V. Sassone, P. Sobociński. Reactive systems over cospans. In *Proc. of LICS '05*. Pp. 311–320. IEEE, 2005.