A Logic on Subobjects and Recognizability

H. J. Sander Bruggink and Barbara König

Universität Duisburg-Essen, Germany
{sander.bruggink,barbara_koenig}@uni-due.de

Abstract. We introduce a simple logic that allows to quantify over the subobjects of a categorical object. We subsequently show that, for the category of graphs, this logic is equally expressive as second-order monadic graph logic (MSOGL). Furthermore we show that for the more general setting of hereditary pushout categories, a class of categories closely related to adhesive categories, we can recover Courcelle's result that every MSOGL-expressible property is recognizable. This is done by giving an inductive translation of formulas of our logic into so-called automaton functors which accept recognizable languages of cospans.

1 Introduction

Regular languages have been studied extensively in computer science and have a large number of applications, such as model checking [3] and termination analysis [10]. The notions of regularity and finite automata can be straightforwardly generalized to trees and tree automata, opening the possibility to define regular tree languages and exploit the convenient closure properties that these languages enjoy. In recent years, the success of regular languages has sparked interest in obtaining a similar notion for other classes of object, in particular graphs. Courcelle has focused on the notion of recognizability – which is equivalent to regularity in the case of word languages – in an algebra of graphs with interfaces. It turns out that recognizable graph languages in this setting can be characterized by locally finite congruences. Bozapalidis and Kalampakas explored a similar characterization based on magmoids [4]. The authors of this paper defined automaton functors to investigate recognizability in a more category theoretic setting [6].

A common disadvantage of the approaches above, is that it is in general not possible to describe a recognizable graph language in a finite way, because the size of the interface of a graph is in principle unbounded. Several solutions for this problem have been developed, such as monadic second-order graph logic [7] and graph automata [5]. In both cases, the class of languages described is a proper subclass of the class of all recognizable graph languages.

In this paper, we develop a logic, similar to monadic second-order graph logic, which describes properties of objects in a category. We show that – under some assumptions on the underlying category – every language describable by the logic is recognizable (the converse does not hold) and that in the category of graphs our logic has the same expressive power as Courcelle's monadic second-order graph logic. This work extends the work of Courcelle in two aspects: First, we generalize the monadic second-order graph logic of Courcelle to arbitrary categories. Second, we prove, for hereditary pushout categories in which for each composable pair of arrows there exist finitely many pushout complements (up to isomorphism), that each language described by a logic formula is recognizable. We do this by giving an inductive construction (on the structure of the formula), which is more convenient in practice than the construction given by Courcelle in [7]. However, another inductive construction has recently been developed independently from our work [9].

The paper is organized as follows: In §2 we give preliminary definitions and fix notation. In §3 we present the syntax and semantics of the logic on subobjects. We continue in §4 to compare the logic to monadic second-order graph logic, and show that, in the category of hypergraphs, the expressive power of our logic and monadic second-order logic are the same. Finally, in §5, we show, by constructing automaton functors from logic formulas, that all languages definable by the logic on subobjects are recognizable, and in §6 an example of the translation is given.

2 Preliminaries

We assume a basic familiarity with category theory. In the following we fix a category $\mathbf{C}$. For a morphism $f\colon A \to B$ we denote by $dom(f) = A$ the domain and by $cod(f) = B$ the codomain of f. When f and g are (composable) morphisms, we write $f\,;\,g$ for the morphism f postcomposed with g, that is $f\,;\,g = g \circ f$.

Let $f\colon A \rightarrowtail T$ and $g\colon B \rightarrowtail T$ be monos with the same codomain T. We write $f \sqsubseteq g$ if there exists an arrow $h\colon A \to B$ (which is necessarily unique) such that $h\,;\,f = g$. The subobject lattice $Sub(T)$ of an object T is formed by isomorphism classes of monos with codomain T, where $\sqsubseteq$ forms the inclusion order. (In practice, we will take unique representatives of the isomorphism classes.)

In the second half of this paper, we will restrict our attention to so-called *hereditary pushout categories* (HPCs) [13], which are related to the well-known adhesive categories [15]. Most adhesive categories are also HPC, including all topoi. In particular, the categories **Set** of sets and **Graph** of hypergraphs (see page 4) are HPC. A category $\mathbf{C}$ is a HPC if

1. $\mathbf{C}$ has pushouts along monos;
2. $\mathbf{C}$ has all pullbacks;
3. given a cube diagram as shown on the right, where a, b and c are monos, the bottom face is a pushout and the back faces are pullbacks, we have that the top face is a pushout if and only if the front faces are pullbacks and d is a mono.

$$\begin{tikzcd}
 & & A' \arrow[dll] \arrow[dr] \arrow[ddd, tail, "a"'] & \\
B' \arrow[dr] \arrow[ddd, tail, "b"] & & & C' \arrow[dll] \arrow[ddd, tail, "c"'] \\
 & D' \arrow[ddd, tail, "d"] & & \\
 & & A \arrow[dll] \arrow[dr] & \\
B \arrow[dr] & & & C \arrow[dll] \\
 & D & &
\end{tikzcd}$$

Different from adhesive categories the vertical arrows in the cube must be mono, instead of one of the arrows in the lower square.

In [6], the authors of the present paper defined recognizable languages of arrows by means of *finite automaton functors*[1] (compare also with a similar notion introduced by Griffing [11]). Let **Rel** be the category which has sets as objects and relations between sets as arrows.

Definition 2.1 (Recognizability). *Let* $\mathbf{C}$ *be a category. An* automaton functor *is a functor* $\mathcal{A}\colon \mathbf{C} \to \mathbf{Rel}$ *which maps each object* X *of* $\mathbf{C}$ *is to a set* $\mathcal{A}(X)$ *(called the set of* states *of* X*) and each arrow* $f\colon X \to Y$ *to a relation* $\mathcal{A}(f) \subseteq \mathcal{A}(X) \times \mathcal{A}(Y)$*. Additionally, each state set* $\mathcal{A}(X)$ *contains a distinguished set of* start states *and a distinguished set of* final states *as subsets.*

An automaton functor is finite *if every set in the image of* $\mathcal{A}$ *is finite, and* deterministic *if every relation* $\mathcal{A}(f)$ *is functional and every state set contains exactly one initial state.*

Let J, K *be two* $\mathbf{C}$*-objects. The* (J, K)-language $L_{J,K}(\mathcal{A})$ *(of arrows from* J *to* K*) is defined as follows:* $f\colon J \to K$ *is contained in* $L_{J,K}(\mathcal{A})$ *if and only if* $\mathcal{A}(f)$ *relates a start state of* $\mathcal{A}(J)$ *to a final state of* $\mathcal{A}(K)$*.*

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

This notion of recognizable language is a generalization of finite automata for word languages. If we take $\mathbf{C}$ as a one-object category with all words as arrows (the free monoid of the alphabet), then a finite automaton is isomorphic to an automaton functor (mapping the single object to the state set of the automaton and each arrow to its respective transition relation).

The intuition behind the definition is to have a mapping into a (finite) domain that respects compositionality and identities, that is, which is a functor. The functor property ensures that decomposing the arrow in different ways does not affect acceptance in any way. This is different from the case of words where there is essentially only one way to decompose a word into atomic components.

Let $\mathbf{C}$ be a category with pushouts. A cospan $c\colon D \xrightarrow{c_L} E \xleftarrow{c_R} F$ is a pair of $\mathbf{C}$-arrows with the same codomain. Here, D and F 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 $c\colon D \xrightarrow{c_L} E \xleftarrow{c_R} F$ and $d\colon F \xrightarrow{d_L} G \xleftarrow{d_R} H$ 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. (In practice, we will choose unique representatives from each isomorphism class.)

Now, the category $Cospan(\mathbf{C})$ is defined as the category which has the objects of $\mathbf{C}$ as objects, and semi-abstract cospans as arrows. In [6] we have shown that Courcelle's notion of recognizable graph language [7, 8] coincides with our notion of recognizability in the category of cospans of graphs when we consider cospans of the form $\emptyset \to G \leftarrow \emptyset$.

[1] What we call *finite automaton functor* here, is simply called *automaton functor* in [6].

For the comparison with the monadic second-order logic of Courcelle, and for most examples, we introduce a category of hypergraphs (just called *graphs* in the following). Fix a signature Σ of labels, each element A of which has an arity $ar(A)$. A graph is a tuple $G = \langle V_G, E_G, att_G, lab_G \rangle$, consisting of a set V_G of vertices (or nodes), a set E_G of edges, an attachment function $att_G \colon E_G \to V_G^*$ and a labeling function $lab_G \colon E_G \to \Sigma$, such that for each $e \in E_G$ it holds that $|att_G(e)| = ar(lab(e))$. (Here, A^* is the set of finite sequences over A, and $|\boldsymbol{a}|$ denotes the length of a sequence $\boldsymbol{a}$.) A graph is *discrete* if it has no edges. A graph morphism is a structure preserving map between graphs. The category **Graph** is the category of finite graphs and graph morphisms.

We define the following "special" graphs and morphisms: The graph Dis_k is the discrete graph with node set $\{1, \ldots, k\}$. Furthermore, for each $A \in \Sigma$, we define the graph E_A, consisting of a single A-labelled edge and adjacent (pairwise unequal) nodes, and the morphisms $e_A^i \colon Dis_1 \to E_A$ mapping the single node in Dis_1 to the node connected to the i-th port of the single edge $e \in E_A$ (that is, to the ith node of $att_{E_A}(e)$). Furthermore, we define Epi_A to be the set of epimorphisms with domain E_A, up to isomorphism (in the case of **Graph** this set is finite).

For the examples, we will usually consider unlabeled, directed (multi)graphs, i.e. we take take $\Sigma = \{\star\}$, with $ar(\star) = 2$. We define $E = E_\star$ to be the graph consisting of a single (directed) edge connecting two nodes, and morphisms $src = e_\star^0$ and $tgt = e_\star^1$, mapping the node of Dis_1 to the source and the target of the edge in E, respectively.

3 A Logic on Subobjects

In this section we introduce the syntax and semantics of the logic for a fixed category $\mathbf{C}$. The logic will be used to describe properties of objects of $\mathbf{C}$.

Syntax. Let Var be a (countably infinite) set of variables. A variable typing is a partial map $\tau \colon Var \rightharpoonup Obj(\mathbf{C}) \cup \{\Omega\}$. There are two kinds of variables: *first-order variables* of sort T (where T is an arbitrary object of $\mathbf{C}$), representing subobjects of fixed structure, and *second-order variables* of sort Ω, representing arbitrary subobjects.[2] Unless otherwise indicated, first-order variables are denoted by lowercase letters (x, y, z) and second-order variables by capitals (X, Y, Z).

Subobject *expressions* are generated by the grammar $e := X \mid f \mathbin{;} x$, where X is of sort Ω, x is of sort T and $f \colon T' \rightarrowtail T$ is a mono. We then say that the expression $f \mathbin{;} x$ is of sort T'. (Intuitively f restricts the graph denoted by x to a subgraph T'.) Variables of sort Ω cannot be precomposed with monos.

The set $Form(\tau)$ of formulas typed by τ is specified by the following grammar:

$$Form(\tau) := e_1 \sqsubseteq e_2 \mid Form(\tau) \wedge Form(\tau) \mid \neg Form(\tau) \mid$$
$$(\exists X \colon \Omega)\, Form(\tau[X \mapsto \Omega]) \mid (\exists x \colon T)\, Form(\tau[x \mapsto T]).^{3}$$

[2] Later, in the comparison with MSOGL, it will become clearer why these types of variables are called first-order and second-order, respectively.

Note that in a formula of the form $e_1 \sqsubseteq e_2$ the two expressions can have arbitrary sorts. The set of free variables of a formula φ, denoted $FV(\varphi)$ is defined as usual. We also define the abbreviation $(x = y) \equiv (x \sqsubseteq y \wedge y \sqsubseteq x)$. Furthermore, we use the usual abbreviations for falsity ($\bot$), disjunction ($\vee$), implication ($\rightarrow$) and universal quantification ($\forall$).

Note that we do not define syntax for defining the morphism f and the object T in expressions and formulas of the form $f \mathbin{⨟} x$ and $(\exists\varphi\colon T)$. The exact syntax needed for this depends on the category, and falls outside the scope of this paper.

Semantics. Let φ be formula typed by variable typing τ and B a $\mathbf{C}$-object. A B-valuation η for φ is a function which assigns:

- to each $x \in FV(\varphi)$, with $\tau(x) = T$ (where $T \neq \Omega$), a mono $v_x\colon T \rightarrowtail B$; and
- to each $X \in FV(\varphi)$, with $\tau(X) = \Omega$, a mono $v_X\colon V \rightarrowtail B$ (where V is arbitrary).

Now we can define the semantics of formulas of the logic. Let φ be a subobject formula typed by τ and let B be a $\mathbf{C}$-object. We say that $B, \eta \models \varphi$, for some B-valuation η, whenever:

- $B, \eta \models X \sqsubseteq Y$ if $\eta(X) \sqsubseteq \eta(Y)$.
- $B, \eta \models f \mathbin{⨟} x \sqsubseteq g \mathbin{⨟} y$ if $f\ ;\ \eta(x) \sqsubseteq g\ ;\ \eta(y)$.
- $B, \eta \models f \mathbin{⨟} x \sqsubseteq Y$ if $f\ ;\ \eta(x) \sqsubseteq \eta(Y)$.
- $B, \eta \models X \sqsubseteq g \mathbin{⨟} y$ if $\eta(X) \sqsubseteq g\ ;\ \eta(y)$.
- $B, \eta \models \varphi_1 \wedge \varphi_2$ if $B, \eta \models \varphi_1$ and $B, \eta \models \varphi_2$.
- $B, \eta \models \neg\varphi$ if $B, \eta \not\models \varphi$.
- $B, \eta \models (\exists x\colon T)\,\varphi$ if there is a mono $v\colon T \rightarrowtail D$ such that $B, \eta[x \mapsto v] \models \varphi$.
- $B, \eta \models (\exists X\colon \Omega)\,\varphi$ if there is a mono $v\colon V \rightarrowtail D$ such that $B, \eta[X \mapsto v] \models \varphi$.

Furthermore, for a closed formula φ, we write $B \models \varphi$ whenever $B, \eta \models \varphi$ for the empty valuation η. Note that this definition works for any category. However the results of §5 (translation of formulas into automaton functors) will only be valid for hereditary pushout categories (which satisfy some additional conditions).

Examples. In an arbitrary category $\mathbf{C}$ we can define the following formula:

- The join of two expressions:
  $(e = e_1 \sqcup e_2) := e_1 \sqsubseteq e \wedge e_2 \sqsubseteq e \wedge (\forall X\colon \Omega)\,\big((e_1 \sqsubseteq X \wedge e_2 \sqsubseteq X) \rightarrow e \sqsubseteq X\big)$.

In the next examples we will use the category of unlabeled directed graphs, as presented on page 4.

[3] Let $f\colon A \rightarrow B$ be a function and let a, b be two elements, which are not necessarily contained in A or B. Then $f[a \mapsto b]\colon A \cup \{a\} \rightarrow B \cup \{b\}$ denotes the function defined as follows:

$$f[a \mapsto b](a') = \begin{cases} b & \text{if } a' = a \\ f(a') & \text{otherwise} \end{cases}$$

- The subgraph X is closed under reachability:
  $RC(X\colon \Omega) := (\forall y\colon E)\,(src \mathbin{;} y \sqsubseteq X \rightarrow tgt \mathbin{;} y \sqsubseteq X)$
- There exists a path from node x to node y (every reachability closed subgraph containing x also contains y):
  $Path(x, y\colon Dis_1) := (\forall Z\colon \Omega)\,\big((id \mathbin{;} x \sqsubseteq Z \wedge RC(Z)) \rightarrow id \mathbin{;} y \sqsubseteq Z\big)$

4 Comparison to Monadic Second-Order Graph Logic

Consider the category **Graph** as presented on page 4. We show that for this category the logic on subobjects has the same expressive power as monadic second-order graph logic [8, 7]. We do this by defining translations from monadic second-order logic to the logic on subobjects and vice versa, and proving that a graph satisfies a formula if and only if it satisfies the translation of the formula.

4.1 Monadic Second-Order Graph Logic

First we define monadic second-order graph logic (MSOGL), mainly in order to fix notation and terminology. This logic is one of the most important specification logics for graphs. Especially, Courcelle's theorem says that every graph property definable in MSOGL is decidable in linear time on (finite) graphs of bounded tree-width.

The MSOGL is a sorted second-order logic with four kinds of variables: first-order node variables (range over nodes), first-order edge variables (range over edges), second-order node variables (range over sets of nodes) and second-order edge variables (range over sets of edges). As a notational convention, first-order variables will be denoted by lowercase letters (x, y, z) and second-order variables by capitals (X, Y, Z). The syntax of MSOGL is given by the following grammar:

$$\begin{aligned}\varphi := \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid (\exists X\colon V)\,\varphi \mid (\exists X\colon E)\,\varphi \mid (\exists x\colon v)\,\varphi \mid (\exists x\colon e)\,\varphi \mid \\ x = y \mid x \in X \mid edge_A(x, y_1, \ldots, y_{ar(A)}),\end{aligned}$$

where typing must be respected, that is, in formulas of the form $x = y$ both variables have the same type and in formulas of the form $x \in X$ it holds that x is a first-order node (edge) variable and X a second-order node (edge) variable. Formulas of the form $edge_A(x, y_1, \ldots, y_{ar(A)})$ denote that the edge x has label A and is adjacent to the nodes $y_1, \ldots, y_n$.

A graph $G = \langle V_G, E_G, att_G, lab_G\rangle$ satisfies a formula φ, written $G \models \varphi$, if there exists a valuation θ, mapping first-order variables to nodes and edges of G and second-order variables to sets of nodes and sets of edges of G, such that $G, \theta \models \varphi$, where:

- $G, \theta \models x = y$ if $\theta(x) = \theta(y)$ and $G, \theta \models x \in X$ if $\theta(x) \in \theta(X)$.
- $G, \theta \models edge_A(x, y_1, \ldots, y_n)$ if $lab_G(\theta(x)) = A$ and $att_G(\theta(x)) = \theta(y_1) \ldots \theta(y_n)$.
- $G, \theta \models \varphi_1 \wedge \varphi_2$ if $G, \theta \models \varphi_1$ and $G, \theta \models \varphi_2$.
- $G, \theta \models \neg\varphi$ if $G, \theta \not\models \varphi$.
- $G, \theta \models (\exists x\colon v)\,\varphi$ if there is a $v' \in V_G$ such that $G, \theta[x \mapsto v'] \models \varphi$.

- $G, \theta \models (\exists x\colon e)\, \varphi$ if there is a $e' \in E_G$ such that $G, \theta[x \mapsto e'] \models \varphi$.
- $G, \theta \models (\exists X\colon V)\, \varphi$ if there is a $V \subseteq V_G$ such that $G, \theta[X \mapsto V] \models \varphi$.
- $G, \theta \models (\exists X\colon E)\, \varphi$ if there is a $E \subseteq E_G$ such that $G, \theta[X \mapsto E] \models \varphi$.

In [7] an extension to monadic second-order logic is presented which also considers cardinality constraints of the form $card_{n,p}(X)$, expressing that the set represented by X contains n elements modulo p, which are omitted here for simplicity. It is currently not entirely clear to us how to integrate a similar predicate into the logic on subobjects in a natural way.

4.2 From Monadic Second-Order Logic to the Logic on Subobjects

We define a translation $[\![\cdot]\!]_{\mathrm{S}}$ from formulas of MSOGL to formulas of the logic on subobjects. For this, we define an edge typing as a map ζ from first-order edge variables (labelled with A) to epimorphisms (with domain E_A). We define the translation function $[\![\varphi]\!]_{\mathrm{S}} = [\![\varphi]\!]_{\mathrm{S}}^{\emptyset}$ where, for an edge typing ζ, $[\![\varphi]\!]_{\mathrm{S}}^{\zeta}$ is inductively defined as follows:

$$\begin{aligned}
[\![\neg\varphi]\!]_{\mathrm{S}}^{\zeta} &:= \neg[\![\varphi]\!]_{\mathrm{S}}^{\zeta} & [\![x = y]\!]_{\mathrm{S}}^{\zeta} &:= x = y \\
[\![\varphi \wedge \psi]\!]_{\mathrm{S}}^{\zeta} &:= [\![\varphi]\!]_{\mathrm{S}}^{\zeta} \wedge [\![\psi]\!]_{\mathrm{S}}^{\zeta} & [\![x \in X]\!]_{\mathrm{S}}^{\zeta} &:= x \sqsubseteq X \\
[\![(\exists X\colon V)\, \varphi]\!]_{\mathrm{S}}^{\zeta} &:= (\exists X\colon \Omega)\, [\![\varphi]\!]_{\mathrm{S}}^{\zeta} & [\![(\exists x\colon v)\, \varphi]\!]_{\mathrm{S}}^{\zeta} &:= (\exists x\colon Dis_1)\, [\![\varphi]\!]_{\mathrm{S}}^{\zeta} \\
[\![(\exists X\colon E)\, \varphi]\!]_{\mathrm{S}}^{\zeta} &:= (\exists X\colon \Omega)\, [\![\varphi]\!]_{\mathrm{S}}^{\zeta} & [\![(\exists x\colon e)\, \varphi]\!]_{\mathrm{S}}^{\zeta} &:= \bigvee_{A \in \zeta} \bigvee_{f \in Epi_A} (\exists x\colon cod(f))\, [\![\varphi]\!]_{\mathrm{S}}^{\zeta[x \mapsto f]}
\end{aligned}$$

$$[\![edge_A(x, y_1, \ldots, y_n)]\!]_{\mathrm{S}}^{\zeta} := \begin{cases} \bigwedge_{1 \le i \le ar(A)} (e_A^i \,;\, \zeta(x)) \mathbin{⨾} x = y_i & \text{if } dom(\zeta(x)) = E_A \\ \bot & \text{otherwise.} \end{cases}$$

Proposition 4.1. *Let G be a graph, and φ a closed formula of monadic second order logic. Then $G \models_{\mathrm{M}} \varphi$ if and only if $G \models [\![\varphi]\!]_{\mathrm{S}}$.*

4.3 From the Logic on Subobjects to Monadic Second-Order Logic

We define a translation $[\![\cdot]\!]_{\mathrm{M}}$ from formulas of our logic to formulas of MSOGL. The main difference between the two logics is that in our logic, we can quantify over arbitrary subobjects, while in MSOGL we can only quantify over nodes, edges, sets of nodes and sets of edges. Thus, a single quantification in the logic on subobjects will in general correspond to more than one quantification in MSOGL. In order to make sure that the multiple variables in an MSOGL-formula evaluate to a possible subobject, we need to express the following two "consistency properties" as MSOGL-formulas (the exact definitions of both formulas is left as an exercise to the reader):

- Let T be a graph and $f\colon X \to (V_T \cup E_T)$ a bijection between first-order variables and nodes and edges of T. The formula $struct_f(T)$ expresses that the nodes and edges assigned to the variables in the codomain of f build a graph isomorphic to T.

– Let X_V be a second-order node variable and X_E a second-order edge variable. The formula $cons(X_E, X_V)$ expresses that all nodes adjacent to an edge in X_E must be in X_V.

A *variable mapping* ξ is a function which maps:

– each free first-order variable x of type T in the formula over subobjects to a bijection from a subset of first-order node and edge variables in the MSOGL-formula to the nodes and edges of T, and
– each free second-order variable X of the source formula to a pair of a second-order node and a second-order edge variable in the target formula.

Now, we define $[\![\varphi]\!]_{\mathrm{M}} = [\![\varphi]\!]_{\mathrm{M}}^{\emptyset}$, where $[\![\varphi]\!]_{\mathrm{M}}^{\xi}$, parametrized by a variable mapping ξ, is inductively defined as follows:

$$\begin{aligned}
[\![\neg\varphi]\!]_{\mathrm{M}}^{\xi} &:= \neg[\![\varphi]\!]_{\mathrm{M}}^{\xi}\\
[\![\varphi \wedge \psi]\!]_{\mathrm{M}}^{\xi} &:= [\![\varphi]\!]_{\mathrm{M}}^{\xi} \wedge [\![\psi]\!]_{\mathrm{M}}^{\xi}\\
[\![(\exists X\colon \Omega)\, \varphi]\!]_{\mathrm{M}}^{\xi} &:= (\exists X_E\colon E)\, (\exists X_V\colon V)\; cons(X_E, X_V) \wedge [\![\varphi]\!]_{\mathrm{M}}^{\xi[X\mapsto\langle X_E, X_V\rangle]}\\
&\ \text{where } X_E \text{ and } X_V \text{ are fresh variables.}\\
[\![(\exists x\colon T)\, \varphi]\!]_{\mathrm{M}}^{\xi} &:= (\exists x_1,\ldots,x_m\colon e)\, (\exists y_1,\ldots,y_n\colon v)\; struct_{\xi'(x)}(T) \wedge [\![\varphi]\!]_{\mathrm{M}}^{\xi'}\\
&\ \text{where } T \text{ is a graph with } m \text{ edges } (e_1,\ldots,e_m) \text{ and } n \text{ nodes}\\
&\ (v_1,\ldots,v_n)\text{, the } x_i \text{ and } y_i \text{ are fresh variables, and}\\
&\ \xi' := \xi[x \mapsto \{x_1 \mapsto e_1,\ldots,x_m \mapsto e_m, y_1 \mapsto v_1,\ldots,y_n \mapsto v_n\}].
\end{aligned}$$

$$\begin{aligned}
[\![X \sqsubseteq Y]\!]_{\mathrm{M}}^{\xi} &:= (\forall x\colon e)\, \big(x \in X_E \to x \in Y_E\big) \wedge (\forall x\colon v)\, \big(x \in X_V \to x \in Y_V\big)\\
&\ \text{where } (X_E, X_V) := \xi(X) \text{ and } (Y_E, Y_V) := \xi(Y).\\
[\![f \mathbin{;} x \sqsubseteq X]\!]_{\mathrm{M}}^{\xi} &:= x_1 \in X_E \wedge \cdots \wedge x_m \in X_E \wedge y_1 \in X_V \wedge \cdots \wedge y_n \in X_V\\
&\ \text{where } (X_E, X_V) := \xi(X),\\
&\ \qquad \{x_1,\ldots,x_m\} := \{u \mid \exists e\colon \langle u, f(e)\rangle \in \xi(x)\} \text{ and}\\
&\ \qquad \{y_1,\ldots,y_n\} := \{w \mid \exists v\colon \langle w, f(v)\rangle \in \xi(x)\}.\\
[\![X \sqsubseteq f \mathbin{;} x]\!]_{\mathrm{M}}^{\xi} &:= (\forall x'\colon e)\, \big(x' \in X_E \to (x' = x_1 \vee \cdots \vee x' = x_m)\big) \wedge\\
&\ (\forall y'\colon v)\, \big(y' \in X_V \to (y' = y_1 \vee \cdots \vee y' = y_n)\big)\\
&\ \text{where } (X_E, X_V) := \xi(X),\\
&\ \qquad \{x_1,\ldots,x_m\} := \{u \mid \exists e\colon \langle u, f(e)\rangle \in \xi(x)\} \text{ and}\\
&\ \qquad \{y_1,\ldots,y_n\} := \{w \mid \exists v\colon \langle w, f(v)\rangle \in \xi(x)\}.\\
[\![f \mathbin{;} x \sqsubseteq g \mathbin{;} x']\!]_{\mathrm{M}}^{\xi} &:= (x_1 = x'_1 \vee \cdots \vee x_1 = x'_p) \wedge \cdots \wedge (x_m = x'_1 \vee \cdots \vee x_m = x'_p) \wedge\\
&\ (y_1 = y'_1 \vee \cdots \vee y_1 = y'_q) \wedge \cdots \wedge (y_n = y'_1 \vee \cdots \vee y_n = y'_q)\\
&\ \text{where } \{x_1,\ldots,x_m\} := \{u \mid \exists e\colon \langle u, f(e)\rangle \in \xi(x)\},\\
&\ \qquad \{y_1,\ldots,y_n\} := \{w \mid \exists v\colon \langle w, f(v)\rangle \in \xi(x)\},\\
&\ \qquad \{x'_1,\ldots,x'_p\} := \{u \mid \exists e\colon \langle u, f(e)\rangle \in \xi(x')\} \text{ and}\\
&\ \qquad \{y'_1,\ldots,y'_q\} := \{w \mid \exists v\colon \langle w, f(v)\rangle \in \xi(x')\}.
\end{aligned}$$

Proposition 4.2. *Let G be a graph, and φ a closed formula of the logic on subobjects. Then $G \models \varphi$ if and only if $G \models_{\mathrm{M}} [\![\varphi]\!]_{\mathrm{M}}$.*

5 Logic and Recognizability

In this section we prove a generalization of Courcelle's result which says that every language definable in monadic second-order graph logic is recognizable [7, 8]: we show that every language of objects of a hereditary pushout category definable by our logic is recognizable in the sense described in [6], by giving an encoding of a formula into an automaton functor. The resulting automaton functor is finite if the category enjoys the property that each composable pair of arrows has finitely many pushout complements (up to isomorphism). In order to be able to use structural induction on the logical formula, we need objects which keep track of the free variables. To this end we introduce a category of objects with valuations.

We fix a hereditary pushout category $\mathbf{C}$ with an initial object 0.[4]

Definition 5.1 (Object with valuation). *Let τ be a variable typing. An* object with valuation *(short:* V-object*) of type τ is a triple $D = \langle \mathrm{B}^D, \eta^D, \sigma^D \rangle$, where*

- B^D *(the* base object*) is a $\mathbf{C}$-object;*
- η^D *maps each variable x for which $\tau(x)$ is defined to a mono $\eta^D(x)\colon V_x^D \rightarrowtail \mathrm{B}^D$ (the* valuation morphism*); and*
- σ^D *maps each variable x for which $\tau(x)$ is defined and $\tau(x) \neq \Omega$ to an arrow $\sigma^D(x)\colon V_x^D \to \tau(x)$ (the* typing morphism*).*

We say that D is well-typed *whenever all arrows $\sigma^D(x)$ are identities.*

Let τ be defined only on the variables $x_1, \ldots, x_n$ and $Y_1, \ldots, Y_m$, with $\tau(x_i) = T_i$ and $\tau(Y_j) = \Omega$ (for $1 \leq i \leq n$ and $1 \leq j \leq m$). Then a V-object of type τ is a diagram as shown on the right (where $t_i = \sigma^D(x_i)$, $v_i = \eta^D(x_i)$ and $w_j = \eta^D(Y_j)$). Note that if D is well-typed (all typing morphisms are identities), η^D corresponds exactly to a valuation (see §3). For the translation we need to allow typing morphisms which are not identities, however, because some objects of the category may not contain the type in its entirety.

$$\begin{tikzcd}
T_1 & \cdots & T_n & & & \\
V_{x_1}^D \arrow[u, "t_1"] \arrow[drrr, tail, "v_1"'] & \cdots & V_{x_n}^D \arrow[u, "t_n"'] \arrow[dr, tail, "v_n"] & V_{Y_1}^D \arrow[d, tail, "w_1"] & \cdots & V_{Y_m}^D \arrow[dll, tail, "w_m"] \\
 & & & \mathrm{B}^D & &
\end{tikzcd}$$

We introduce the following operators for diagram extension and restriction. Let $D = \langle \mathrm{B}^D, \eta^D, \sigma^D \rangle$ be a V-object of type τ. By $\mathit{drop}_x(D)$ we denote the V-object of type $\tau|_{\mathit{Var}\setminus\{x\}}$ from which the morphisms and objects associated to x have been dropped. Furthermore, let $v\colon V \rightarrowtail \mathrm{B}^D$ and $t\colon V \rightarrowtail T$ be given. Then $\mathit{add}_y^v(D) = \langle \mathrm{B}^D, \eta^D[y \mapsto v], \sigma^D \rangle$ denotes the V-object of type $\tau[y \mapsto \Omega]$, which is obtained by adding the arrow v (indexed by y) to D. Similarly, $\mathit{add}_y^{v,t}(D) = \langle \mathrm{B}^D, \eta^D[y \mapsto v], \sigma^D[y \mapsto t] \rangle$ denotes the V-object of type $\tau[y \mapsto T]$ which is obtained by adding the arrows v and t (indexed by y).

[4] We require an initial object since we want to convert an arbitrary object A into the corresponding cospan $0 \to A \leftarrow 0$. Then the automaton functor recognizes such cospans by "starting" and "ending" with the initial object.

We consider the following category of objects with vertical interfaces, some of which might be typed.

Definition 5.2 (Category of objects with valuations). *Let τ be a variable typing. We define the category $\mathbf{V}_\tau^{\mathbf{C}}$ (or simply $\mathbf{V}_\tau$) as follows.*

The objects of $\mathbf{V}_\tau^{\mathbf{C}}$ are the objects with valuation of type τ (see Def. 5.1). An arrow $p\colon D \to E$ of $\mathbf{V}_\tau^{\mathbf{C}}$ consists of $\mathbf{C}$-arrows $\alpha^p\colon \mathrm{B}^D \to \mathrm{B}^E$ and $\nu_x^p\colon \eta^D(x) \to \eta^E(x)$ (for each $x \in Var$ such that $\tau(x)$ is defined) such that the square below is a pullback and the triangle commutes (whenever $\tau(x) \neq \Omega$):

$$\begin{tikzcd}
V_x^D \arrow[r, "\nu_x^p"] \arrow[d, tail, "\eta^D(x)"'] \arrow[dr, phantom, "(\mathrm{PB})"] & V_x^E \arrow[d, tail, "\eta^E(x)"] \\
\mathrm{B}^D \arrow[r, "\alpha^p"] & \mathrm{B}^E
\end{tikzcd}
\qquad
\begin{tikzcd}
V_x^D \arrow[rr, "\nu_x^z"] \arrow[dr, "\sigma^D(x)"'] & & V_x^E \arrow[dl, "\sigma^D(x)"] \\
& \tau(x) &
\end{tikzcd}$$

The two diagrams above specify the following: the valuation morphisms of D can be obtained from the valuation morphisms of E by taking a pullback (if α^p is mono this is simply some form of restriction). Furthermore the typing morphisms must be consistent.

We will consider (semi-abstract) cospans over $\mathbf{V}_\tau^{\mathbf{C}}$, i.e., we will work with the category $Cospan(\mathbf{V}_\tau^{\mathbf{C}})$. Note that due to Property 3 on page 2 pushouts in $\mathbf{V}_\tau^{\mathbf{C}}$ – and hence cospan composition in $Cospan(\mathbf{V}_\tau^{\mathbf{C}})$ – can be computed componentwise by taking pushouts of all the component morphisms and obtaining the morphisms of the resulting diagram as mediating morphisms. In the initial object 0 all objects (apart from the $\tau(x)$) are the initial objects of $\mathbf{C}$. We extend the operation $drop_x$ to $\mathbf{V}_\tau^{\mathbf{C}}$-cospans in the straightforward way.

Given a $\mathbf{V}_\tau$-object $D = \langle \mathrm{B}^D, \eta^D, \sigma^D \rangle$ and a variable x such that $\tau(x)$ is defined, we define the *selection morphism* sel_x^D (in the case that $\tau(x) \neq \Omega$ parametrized by a morphism $f\colon V_x^D \to \tau(x)$) as follows:

$$\begin{tikzcd}
U \arrow[r] \arrow[d, tail, "g"] \arrow[dr, phantom, "(\mathrm{PB})"] \arrow[dd, tail, dashed, bend right=50, "sel_{f,x}^D"'] & S \arrow[d, tail, "f"] \\
V_x^D \arrow[r, "\sigma^D(x)"'] \arrow[d, tail, "\eta^D(x)"] & \tau(x) \\
\mathrm{B}^D &
\end{tikzcd}$$

- Suppose $\tau(x) \neq \Omega$. Let $f\colon S \rightarrowtail \tau(x)$. We take the pullback of $\sigma^D(x)$ and f, and obtain an object U and arrows $U \to S$ and $g\colon U \to V_x^D$ (see the diagram on the right). Now we take $sel_{f,x}^D := g \,;\, v_x^D$.
- Suppose $\tau(x) = \Omega$. Then we simply take $sel_x^D := \eta^D(x)$.

We extend the definition of selection morphism to expressions of subobject logic in the obvious way, i.e. $sel_{f\,⨾\,x}^D := sel_{f,x}^D$.

We will now present an inductive encoding that takes a formula $\varphi \in Form(\tau)$ and translates it into an automaton functor $\mathcal{A}_\tau^\varphi$ (or simply $\mathcal{A}$) for the category $Cospan(\mathbf{V}_\tau^{\mathbf{C}})$. The definition is divided into four subcases: *atomic formulas*, *boolean operations*, *first-order quantification* and *second-order quantification.*

Atomic formulas: Suppose $\varphi \equiv e_1 \sqsubseteq e_2$. Since the variables contained in e_1, e_2 are free and hence the corresponding vertical interfaces are present, it is enough to just check inclusion of those interfaces. This information is recorded in just one state $\star$ and we produce the empty relation whenever inclusion fails.

Formally, we define the automaton functor $\mathcal{A}^{\varphi}_{\tau}$ as follows. To each object D, $\mathcal{A}^{\varphi}_{\tau}$ assigns the one-element set $\{\star\}$ if $sel^{D}_{e_1} \sqsubseteq sel^{D}_{e_2}$ (where $\star$ is the initial as well as the final state), and the empty set otherwise.

For a cospan $c\colon D \xrightarrow{l} E \xleftarrow{r} F$ we define $\mathcal{A}^{\varphi}_{\tau}(c) = \mathcal{C}(l) \mathbin{;} \mathcal{C}(r)^{-1}$, where, for a $\mathbf{V}_{\tau}$-arrow $p\colon D_1 \to D_2$, we define

$$\mathcal{C}(p) := \begin{cases} id_{\{\star\}} & \text{if } sel^{D_2}_{e_1} \sqsubseteq sel^{D_2}_{e_2} \\ \emptyset & \text{otherwise.} \end{cases}$$

Here, $id_{\{\star\}}$ refers to the identity relation on $\{\star\}$ and $\emptyset$ to the empty relation. Note that the empty relation is the identity relation on the empty set, i.e. $id_{\emptyset} = \emptyset$.

Boolean operations: Suppose $\varphi \equiv \varphi_1 \wedge \varphi_2$ or $\varphi \equiv \neg\varphi$. In this case compute the automaton functors for the subformulas and apply the standard techniques for intersection or complement to the automaton functors (due to a result of [6] these operations can be performed on all automaton functors, independent of the category). In the latter case (negation/complement) it is necessary to first construct the equivalent deterministic automaton functor.

Second-order quantification: Since the second-order quantification case is much simpler than the first-order case, we present it first. Suppose $\varphi \equiv (\exists X\colon \Omega)\,\varphi'$. Let $\mathcal{A}' := \mathcal{A}^{\varphi'}_{\tau'}$, where $\tau' = \tau[X \mapsto \Omega]$, be the automaton functor constructed for φ'.

The domain of the automaton functor for the subformula φ' is a category of objects with an additional vertical interface. The automaton functor for φ works in the following way: non-deterministically, a satisfying assignment for X is chosen and then it behaves like the automaton functor for φ'.

The automaton functor $\mathcal{A}$ for φ is formally defined as follows:

$$\mathcal{A}(D) = \bigcup_{v \in Sub(\mathrm{B}^{D})} \mathcal{A}'(add^{v}_{X}(D)) \times \{v\},$$

where $v \in Sub(\mathrm{B}^{D})$ means that v ranges over representatives of isomorphism classes of monos into B^{D}. To a cospan $c\colon D \xrightarrow{l} E \xleftarrow{r} F$ we assign the following relation $\mathcal{A}(c) \subseteq \mathcal{A}(D) \times \mathcal{A}(F)$: $\langle q, v^{D}\rangle$ is in relation with $\langle q', v^{F}\rangle$ whenever there exists a cospan

$$c'\colon add^{v^{D}}_{X}(D) \to E' \leftarrow add^{v^{F}}_{X}(F)$$

of type τ' such that $drop_{X}(c') = c$ and $\langle q, q'\rangle \in \mathcal{A}'(c')$.

Finally, a state $\langle q, v\rangle$ is initial (final) if and only if q is initial (final).

First-order quantification: Suppose $\varphi \equiv (\exists x\colon T)\,\varphi'$. Let $\mathcal{A}' := \mathcal{A}^{\varphi'}_{\tau'}$, where $\tau' = \tau[x \mapsto T]$, be the automaton functor constructed for φ'.

As in the case of second-order quantification, we non-deterministically choose a satisfying assignment for x. This time, however, we have to track how much of the sought after subobject (T) has already been recognized (see explanation below).

We define the new automaton functor $\mathcal{A}^\varphi_\tau$ as follows:

$$\mathcal{A}^\varphi_\tau(D) = \bigcup_{\substack{(v\colon V \to \mathrm{B}^D) \in Sub(\mathrm{B}^D),\\ (t\colon V \to T)}} \mathcal{A}'(add_x^{v,t}(D)) \times \{\langle v, t_1, t_2\rangle \mid t = t_1; t_2, \text{POC exists}\},$$

where, by "POC exists" we mean that there exists arrows s_1, s_2 such that $t_1; t_2 = s_1; s_2$ and the four arrows form a pushout. In the cases of v and the decomposition t_1 ; t_2 we actually take representatives of the respective isomorphism classes.

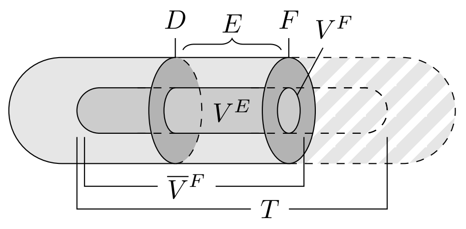
To a cospan $c\colon D \xrightarrow{l} E \xleftarrow{r} F$ we assign the following relation $\mathcal{A}^\varphi_\tau(c) \subseteq \mathcal{A}^\varphi_\tau(D) \times \mathcal{A}^\varphi_\tau(F)$: $\langle q, v^D, t^D_1, t^D_2\rangle$ is in relation with $\langle q', v^F, t^F_1, t^F_2\rangle$ whenever c can be extended to a cospan

$$c'\colon add_x^{v^D,(t_1^D;t_2^D)}(D) \xrightarrow{l'} add_x^{v^E,t^E}(E) \xleftarrow{r'} add_x^{v^F,(t_1^F;t_2^F)}(F),$$

satisfying the following conditions:

– $\langle q, q'\rangle \in \mathcal{A}'(c')$ *and*
– we have the commuting diagram below (on the left), where $\overline{V}^F$ is the pushout object of t^D_1 and $\nu^{l'}$.

$$\begin{tikzcd}
 & & T & & \\
 & \overline{V}^D \arrow[ur, "t_2^D"] \arrow[rr, dashed] & & \overline{V}^F \arrow[ul, "t_2^F"'] & \\
V^D \arrow[ur, "t_1^D"] \arrow[rr, "\nu^{l'}"] \arrow[d, tail, "v^D"'] & & V^E \arrow[uu, "t^E"] \arrow[ur, dashed] \arrow[d, tail, "v^E"'] & & V^F \arrow[ul, "t_1^F"'] \arrow[ll, "\nu^{r'}"'] \arrow[d, tail, "v^F"'] \\
\mathrm{B}^D \arrow[rr, "\alpha^{l'}"] & & \mathrm{B}^E & & \mathrm{B}^F \arrow[ll, "\alpha^{r'}"']
\end{tikzcd}$$



A state $\langle q, v, t_1, t_2\rangle$ is initial if and only if q is initial and t_1 is an identity. It is final if and only if q is final and the arrow t_2 is an identity.

The intuition behind the diagram above is the following: we are tracking an object of type T, while we are reading the "complete" object step by step. The vertical interface V^F (with mono v^F) denotes the part of the outer interface that corresponds to this object. On the other hand $\overline{V}^F$ (with mapping t^F_2 into the type T) tells which part of the object we have already seen. Finally t^F_1 relates the part which is currently in the (outer) interface to the part we have already seen (see image above, on the right). Similar explanations can be given for v^D, t^D_1 and t^D_2. Finally, $\overline{V}^F$ is obtained from $\overline{V}^D$ by gluing those (new) parts of the tracked object that have been seen in the current cospan, namely V^E. (Additional intuition is provided by the example in §6).

Proposition 5.3. $\mathcal{A}^\varphi_\tau$*, constructed for a formula* φ *and a typing* τ *as described above, is a functor, that is, it preserves identities and composition.*

Theorem 5.4. *Let φ be a formula, let τ be a typing and let $D = \langle \mathrm{B}^D, \eta^D, \sigma^D \rangle$ be a well-typed V-object of type τ. Then $\mathrm{B}^D, \eta^D \models \varphi$ if and only if D is contained in the $(0,0)$-language of $\mathcal{A}^\varphi_\tau$, that is, if there exists an initial state i_0 and a final state f_0 with $\langle i_0, f_0 \rangle \in \mathcal{A}^\varphi_\tau(c)$, where c is the unique cospan of the form $0 \to D \leftarrow 0$.*

Finiteness. Note that so far we did not impose any restrictions on the finiteness of the state sets. For this we first require that the objects in our category are finite in the sense that they contain only finitely many subobjects. In the case of graphs that would be the finite graphs and in the case of sets the finite sets.

However, in addition we need another requirement: in the encoding of first-order quantification we split a given arrow t into $t = t_1; t_2$ such that all pushout complements for this split exist. In order to guarantee finiteness it is hence also necessary that a given arrow t only admits finitely many such splits up to isomorphism.

It is currently not clear to us how subobject finiteness is related to the condition of having finitely many pushout complement splits. Clearly the latter property implies the former, but the other direction is unclear. In any case, the category **Graph** satisfies both requirements.

6 Detailed example

As an example, we consider (in the category of cospans of unlabeled directed graphs) the translation into an automaton functor of the formula

$$\neg RC(X) \equiv (\exists y\colon E)\bigl(\mathit{src} \mathbin{⨾} y \sqsubseteq X \wedge \neg(\mathit{tgt} \mathbin{⨾} y \sqsubseteq X)\bigr),$$

which expresses that the subgraph X is not closed under reachability.

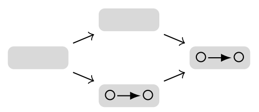
The automaton functor for the atomic formula $\varphi_2 := \mathit{tgt} \mathbin{⨾} y \sqsubseteq X$ maps each graph, in which the target of edge y does not lie in X, to the empty set, and all other graphs to the state set $\{\star\}$. It maps a cospan to $id_{\{\star\}}$ if the above also holds for the middle object of the cospan. The automaton functor of the other atomic formula, $\varphi_1 := \mathit{src} \mathbin{⨾} y \sqsubseteq X$, is built analogously.

To calculate the negation of φ_2, we must first make its automaton functor deterministic by means of the powerset construction (see [6]). Graphs are now mapped to either the state set $\{\emptyset, \{\star\}\}$ (where $\{\star\}$ is initial as well as final) or the state set $\{\emptyset\}$ (no initial or final states), and cospans are mapped accordingly. After the negation, final and non-final states are swapped.

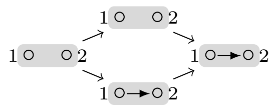
To construct the automaton functor for $\varphi_1 \wedge \neg\varphi_2$, we use a Cartesian product construction. In the rest of the example, we will restrict our attention to an object D' with $\mathcal{A}^{\varphi_1}(D') = \{\star\}$ and $\mathcal{A}^{\neg\varphi_2}(D') = \{\emptyset, \{\star\}\}$. For this object D' we have that $\mathcal{A}^{\varphi_1 \wedge \neg\varphi_2}(D') = \bigl\{\langle\{\star\}, \emptyset\rangle, \langle\{\star\}, \{\star\}\rangle\bigr\}$ where $\langle\{\star\}, \{\star\}\rangle$ is the initial and $\langle\{\star\}, \emptyset\rangle$ the final state. By reaching the final state we record that both the source node of the edge assigned to y is contained in X and the target node is not contained in X.

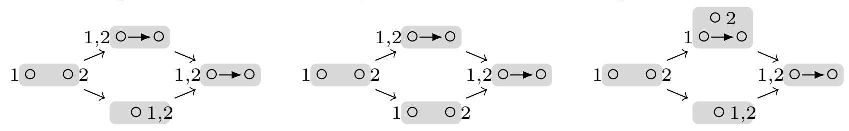
Finally, we build an automaton functor $\mathcal{A}$ for $\varphi \equiv (\exists y\colon E)\,(\varphi_1 \wedge \neg\varphi_2)$. Recall that $E = \circ\!\!\to\!\!\circ$. We obtain a V-object $D = \mathit{drop}_y(D')$. Assume it has base object $\mathrm{B}^D = \mathit{Dis}_2 = \circ \quad \circ$, a two-node discrete graph. States of the new automaton functor consist of arbitrary monos $v\colon V \rightarrowtail \mathrm{B}^D$, allowed decompositions t_1, t_2 of arbitrary typing morphisms $t\colon V \to E$, and the states of D extended with v, t_1, t_2. Let us list the possible decompositions of the typing morphism for all $2^2 = 4$ possible v into B^D:

- For $v\colon \emptyset \rightarrowtail \mathrm{B}^D$, there is a single typing morphism, and two possible decompositions, given by the two legs of the pushout diagram on the right. The top leg expresses the situation where no part of the sought after subobject has been encountered yet, whereas the bottom leg expresses the situation that the entire sought after subobject was already encountered.



- For $v = {}_1\circ \rightarrowtail {}_1\circ \quad \circ$, there are two possible typing morphisms (mapping the node to the source and the target of the edge, respectively), with two decompositions each. For $v = \circ_2 \rightarrowtail \circ \quad \circ_2$, four decompositions are obtained analogously.[5]
- Let $v = id_{\mathrm{B}^D}$. There are four possible typing morphisms. For $t = {}_1\circ \quad \circ_2 \to {}_1\circ\!\!\to\!\!\circ_2$ there are two allowed decompositions, which can be read from the diagram on the right. For $t = {}_1\circ \quad \circ_2 \to {}_{1,2}\circ\!\!\to\!\!\circ$ the allowed decompositions can be read from the pushout diagrams below. Since two decompositions occur twice, there are four decompositions in total.





  For $t = {}_1\circ \quad \circ_2 \to {}_2\circ\!\!\to\!\!\circ_1$ and $t = {}_1\circ \quad \circ_2 \to \circ\!\!\to\!\!\circ_{1,2}$ we symmetrically obtain five decompositions.
  As an example, consider the decomposition ${}_1\circ \quad \circ_2 \to {}_1\circ\!\!\to\!\!\circ \ \circ_2 \to {}_{1,2}\circ\!\!\to\!\!\circ$ (the top right in the three diagrams above). This decomposition expresses that both nodes of B^D are part of the sought after graph (as source node), but still need to be merged.

Suppose that for all possible v, t, the object D extended with v, t has the state set above (2 states). The new automaton functor then has $2 \cdot 20 = 40$ states. Accepting states are the ones which contain an accepting state of the automaton functor for the subterm, and in which the second part of the decomposition (t_2) is an isomorphism (that is, a subgraph isomorphic to E was found).

The automaton functor works as follows: given a cospan $c\colon D \xrightarrow{c_\mathrm{L}} E \xleftarrow{c_\mathrm{R}} F$, it non-deterministically chooses what parts of the new information (the parts of E not in the range of c_L) belong to the sought-after type E, and then works the same as the automaton functor constructed for the subformula.

[5] The numbers beside the nodes indicate which nodes are mapped to which.

7 Conclusion and Further Research

We have introduced a logic on subobjects and shown how it is related to monadic second-order graph logic. Although we are working in a categorical setting our choice was to focus on a classical logic, quantifying over sets of subobjects, and not a categorical logic where the universe is replaced by an object [14, 16]. With our current understanding it is not clear to us how to obtain a similar correspondence of MSOGL with a categorical logic. For instance, our predicates on subobjects can not directly be interpreted as subobjects of a product object, as would be customary for subobject logic. Although our logic falls out of scope of known categorical logics, we still believe that its intimate connection to MSOGL and recognizability makes it an interesting logic to study.

Furthermore it would be interesting to study which kind of equivalences on objects are induced by the logics in various categories (also in **Set**). Another interesting question is to consider in greater detail the relation to the graph predicates of [17] which are equivalent to first-order graph logic.

Note that although here we focused exclusively on graphs as examples, the greater generality of the logic allows us to easily talk about all kinds of "graph-like" structures, such as hierarchical graphs, graphs with scopes, attributed graphs or graphs with higher-order edges.

We also introduced a procedure for translating formulas of our logic inductively into automaton functors, which are automata for accepting cospans in hereditary pushout categories, a class of categories which includes all topoi. That is, we have shown how to convert specifications into algorithms, albeit in a fairly abstract setting. Other methods for converting MSOGL-formulas into recognizable languages which are known to us [7, 12] do not follow such an inductive strategy, but directly specify state sets by forming equivalence classes of logical formulas. We hope that such an inductive method can help in practice in order to generate and use automaton functors. In implementations, we can construct the automaton functor only for a restricted set of "atomic" cospans from which all cospans can be generated by composition. We are especially interested in applications in verification such as invariant checking [1, 2] and termination analysis. Despite the inductive approach the state sets of automata can still become fairly large, as is already evident from the detailed example in §6. Our current approach to solve this problem is to represent automaton functors (which are basically relations) via binary decision diagrams. Initial experiments have been quite encouraging.

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. While we think that this should be feasible in principle, we did not choose to follow this path here since it would have added additional complexity to the encoding into automaton functors. In the implementation we will restrict ourselves to discrete interfaces and, out of necessity, to graphs of bounded pathwidth or treewidth in order to work with finitely many state sets.

Acknowledgements: We are grateful to Tobias Heindel for his suggestions and his help with the categorical foundations. Furthermore we would like to thank Paolo Baldan and Andrea Corradini for stimulating discussions on this research topic.

References

1. C. Blume. Graphsprachen für die Spezifikation von Invarianten bei verteilten und dynamischen Systemen. Master's thesis, Universität Duisburg-Essen, Nov. 2008.
2. Christoph Blume, H.J. Sander Bruggink, and Barbara König. Recognizable graph languages for checking invariants. In *Proc. of GT-VMT '10 (Workshop on Graph Transformation and Visual Modeling Techniques)*, Electronic Communications of the EASST, 2010.
3. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *Proc. of CAV '00*, pages 403–418. Springer, 2000. LNCS 1855.
4. S. Bozapalidis and A. Kalampakas. Recognizability of graph and pattern languages. *Acta Informatica*, 42(8/9):553–581, 2006.
5. S. Bozapalidis and A. Kalampakas. Graph automata. *Theoretical Computer Science*, 393:147–165, 2008.
6. H.J.S. Bruggink and B. König. On the recognizability of arrow and graph languages. In *Proc. of ICGT '08*, pages 336–350. Springer, 2008. LNCS 5214.
7. B. Courcelle. The monadic second-order logic of graphs I. Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.
8. B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1: Foundations*, chapter 5. World Scientific, 1997.
9. Bruno Courcelle and Irène Durand. Verifying monadic second order graph properties with tree automata. In *European Lisp Symposium*, May 2010.
10. A. Geser, D. Hofbauer, and J. Waldmann. Match-bounded string rewriting systems. *Applicable Algebra in Engineering, Communication and Computing*, 15(3–4):149–171, 2004.
11. G. Griffing. Composition-representative subsets. *Theory and Applications of Categories*, 11(19):420–437, 2003.
12. M. Grohe. Logic, graphs and algorithms. In J.Flum, E.Grädel, and T.Wilke, editors, *Logic and Automata – History and Perspectives*. Amsterdam University Press, 2007.
13. T. Heindel. *A category theoretical approach to the concurrent semantics of rewriting*. PhD thesis, Universität Duisburg–Essen, September 2009. http://www.ti.inf.uni-due.de/people/heindel/diss.pdf.
14. Bart Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundation of Mathematics*. Elsevier, 1999.
15. S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *RAIRO – Theoretical Informatics and Applications*, 39(3), 2005.
16. A.M. Pitts. Categorical logic. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science V*. Oxford University Press, 2001.
17. A. Rensink. Representing first-order logic using graphs. In *Proc. of ICGT '04*, pages 319–335. Springer, 2004. LNCS 3256.