Representing First-Order Logic Using Graphs

Arend Rensink

Department of Computer Science, University of Twente
P.O.Box 217, 7500 AE, The Netherlands
rensink@cs.utwente.nl

Abstract. We show how edge-labelled graphs can be used to represent first-order logic formulae. This gives rise to recursively nested structures, in which each level of nesting corresponds to the negation of a set of existentials. The model is a direct generalisation of the *negative application conditions* used in graph rewriting, which count a single level of nesting and are thereby shown to correspond to the fragment $\exists\neg\exists$ of first-order logic. Vice versa, this generalisation may be used to strengthen the notion of application conditions. We then proceed to show how these nested models may be flattened to (sets of) plain graphs, by allowing some structure on the labels. The resulting formulae-as-graphs may form the basis of a unification of the theories of graph transformation and predicate transformation.

1 Introduction

Logic is about expressing and proving constraints on mathematical models. As we know, such constraints can for instance be denoted in a special language, such as First-Order Predicate Logic (FOL); formulae in that language can be interpreted through a satisfaction relation. In this paper we study a different, non-syntactic representation, based on graph theory, which we show to be equivalent to FOL. The advantage of this alternative representation is that it ties up notions from algebraic graph rewriting with logic, with potential benefits to the former.

We start out with the following general observation. A condition that states that a certain sub-structure should exist in a given model can often be naturally expressed by requiring the existence of a *matching* of (a model of) that sub-structure in the model in question. Illustrated on the *edge-labeled graphs* studied in this paper: The existence of entities x and y related by $\mathsf{a}(x, y)$ and $\mathsf{b}(y, x)$ (where a and b are binary relations) can be expressed by the requiring the existence of a matching of the following graph:
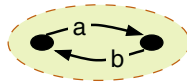

Note that this matching only implies that this sub-structure exists *somewhere* in the target graph; it does not say, for instance, that $\mathsf{a}(x, y)$ and $\mathsf{b}(y, x)$ always go together, or that the entities playing the roles of x and y are unrelated to other entities, or even that these entities are distinct from one another.

One particular context in which matchings of this kind play a prominent role is that of *algebraic graph rewriting* (see [3,8] for an overview). The basic building blocks of

a given graph rewrite system are *production rules*, which (among other things) have so-called "left hand side" (LHS) graphs. A production rule *applies* to a given graph only if the rule's LHS can be matched by that graph;[1] moreover, the effect of the rule application is computed relative to the matching.

The class of conditions that matchings can express is fairly limited. For instance, no kind of *negative* information, such as the absence of relations or the distinctness of x and y in the example above, can be expressed in this manner. In the context of algebraic graph transformation, this observation has led to the definition of so-called *negative application conditions* (NACs) (see [10]). A NAC itself takes the form of a matching of the "base" graph into another. A LHS with NACs, interpreted as a logical constraint, is satisfied if a matching of the LHS exists which, however, cannot be *factored* through any of the NACs.[2] Consider, for instance, the following example of a graph with two NACs:
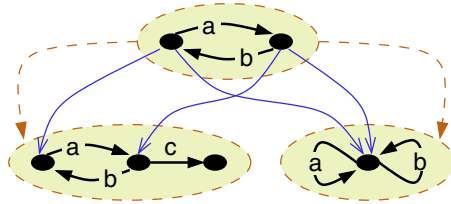

The base graph (here drawn on top) expresses, as before, that there are two entities, say x and y, such that $\mathsf{a}(x, y)$ and $\mathsf{b}(y, x)$; the first NAC adds the requirement that there is no z such that $\mathsf{c}(y, z)$; and the second NAC adds the requirement that x and y are distinct. The combined structure thus represents the formula

$$\exists x, y : \mathsf{a}(x, y) \wedge \mathsf{b}(y, x) \wedge (\nexists z : \mathsf{c}(y, z)) \wedge x \neq y \ .$$

Formally, a graph satisfies this condition if there is a matching of the base graph into it, that cannot be factored through a matching of either of the NAC target graphs.

Although, clearly, NACs strengthen the expressive power of graph conditions, it is equally clear that there are still many properties that can *not* be expressed in this way. For instance, any universally quantified positive formula is outside the scope of NACs. As a very simple example consider $\exists x : \forall y : x = y$ expressing that there exists precisely one entity. However, we can easily add more layers of "application conditions". For instance, the existence of a unique entity is represented by the following structure:

---
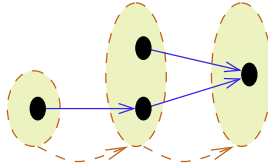
[1] In the single-pushout approach [8], the existence of a matching is also sufficient; in the double-pushout approach [3], there are some further conditions on the applicability of a rule.

[2] An important difference is that, in graph transformation, the issue is not whether a matching of a given LHS exists but to find *all* matchings. Seen in this light, the results of this paper concern the *applicability* of production rules.

At this point, readers familiar with the theory of *existential graphs* (see, e.g., [14,5]) may recognize the analogy between this stacking of graphs and the *cuts* used there to represent negation. See also the discussion of related work in Sect. 6. This paper is devoted to working out the ensuing theory in the category of edge-labeled graphs, providing a direct connection to the setting of algebraic graph transformation. We present the following results:

- Graph conditions with a stack of application conditions of height n, interpreted through matchings as sketched above, are expressively equivalent to a fragment of ($\forall$-free) FOL that may be denoted $\exists(\neg\exists)^n$ — for a precise statement see Th. 4. It is known that a higher value of n gives rise to a truly more expressive fragment of FOL; that is, for each n there is a property that can be expressed in $\exists(\neg\exists)^{n+1}$ and not in $\exists(\neg\exists)^n$. We prove equivalence through compositional translations back and forth.

  As a corollary, NACs carry the expressive power of $\exists\neg\exists$ — which indeed excludes universally quantified positive formulae, since those would translate to $\neg\exists\neg$ at the least. Another consequence is that more highly stacked graph conditions, providing the full power of FOL, can be integrated seamlessly into the theory of algebraic graph transformation. In the conclusions we briefly mention two such extensions that have been studied in the graph transformation literature.
- Graph conditions may be flattened, without loss of information, to simple, edge-labeled graphs, provided we add structure to the labels to reflect the stack height at which the nodes and edges were originally introduced. With hindsight this structure on the labels is indeed strongly reminiscent of the cuts in existential graphs, except that we avoid the need for representing cuts as an explicit structure on the graphs.

The remainder of this paper is structured as follows. In Sect. 2 we recall some definitions. In Sect. 3 we define graph predicates and provide a translation to FOL; in Sect. 4 we define the reverse translation. Sect. 5 shows how to flatten graph predicates. Finally, in Sect. 6 we summarize the results and discuss related work, variations and extensions.

2   Basic Definitions

We assume a countable universe of variables Var, ranged over by x, y, z, and a countable universe of relation (i.e., predicate) symbols Rel $\subseteq$ Lab (not including $=$), ranged over by a. The following grammar defines FOL, the language of first order logic with equality and binary relations:

$$\phi ::= x = x \mid \mathsf{a}(x, x) \mid \neg\phi \mid \bigvee\Phi \mid \bigwedge\Phi \mid \exists X\colon \phi \,.$$

Here $\Phi \subseteq$ FOL and $X \subseteq$ Var are finite sets of formulae and variables, respectively. (So $\exists X\colon \phi$ is not second-order quantification but finitary first-order quantification.) We use $fv(\phi)$ $[fv(\Phi)]$ for $\phi \in$ FOL $[\Phi \subseteq$ FOL$]$ to denote the set of free variables in ϕ $[\Phi]$ (with the usual definition). Note that all sets of free variables are finite.

As models for FOL we use edge-labeled graphs without parallel edges. For this purpose we assume a countable universe of nodes Node, ranged over by v, w, and a countable universe of labels Lab $\supseteq$ Rel, ranged over by ℓ. Except in Sect. 5, we will have Lab $=$ Rel.

Definition 2.1 (graphs).

- *A graph is a tuple $G = \langle N, E \rangle$ where $N \subseteq$ Node is a set of* nodes *and $E \subseteq N \times$ Lab $\times N$ a set of* edges.
- *If G and H are graphs, a* graph morphism *from G to H is a tuple $\mu = (G, H, f)$ where $f : N_G \to N_H$ is such that $(f(v), \ell, f(w)) \in E_H$ for all $(v, \ell, w) \in E_G$.*
- *The* category of graphs, *denoted* Graph, *has graphs as objects and graph morphisms as arrows, with the obvious notions of identity and composition.*

We denote the components of a graph G by N_G and E_G (as already seen above), and we use $\mu : G \to H$ or $\mu \in$ Graph(G, H) to denote that μ is a morphism from G to H. For $\mu : G \to H$ we denote $src(\mu) = G$ and $tgt(\mu) = H$. The following result is standard.

Proposition 1. Graph *is a cartesian closed category with all limits and colimits.*

Every countable set A gives rise to a discrete graph $\langle A \rangle$, with $N = A$ and $E = \emptyset$. We also use $\langle v \overset{\mathsf{a}}{\to} w \rangle$ to denote the one-edge graph with $N = \{v, w\}$ and $E = \{(v, \mathsf{a}, w)\}$. Furthermore, for every $X \subseteq Y \subseteq$ Var (Y countable) we use $emb[X, Y] = (X, Y, id_X)$ for the morphism that embeds $\langle X \rangle$ in $\langle Y \rangle$, and for every $G \in$ Graph we write id_G for the identity morphism on G. The following rules define a modeling relation $\models$ between graph morphisms $\theta \in$ Graph$(\langle X \rangle, G)$ with $X \supseteq fv(\phi)$ (which combine the valuation of the logical variables in X with the algebraic structure of F) and FOL-formulae ϕ:

$$
\begin{aligned}
&\theta \models x = y &&\text{if } \theta(x) = \theta(y) \\
&\theta \models \mathsf{a}(x, y) &&\text{if } (\theta(x), \mathsf{a}, \theta(y)) \in E_G \\
&\theta \models \neg\phi &&\text{if } \theta \not\models \phi \\
&\theta \models \bigvee \Phi &&\text{if } \theta \models \phi \text{ for some } \phi \in \Phi \\
&\theta \models \bigwedge \Phi &&\text{if } \theta \models \phi \text{ for all } \phi \in \Phi \\
&\theta \models \exists Y : \phi &&\text{if } \eta \models \phi \text{ such that } \theta = \eta \circ emb[X, X \cup Y] \ .
\end{aligned}
$$

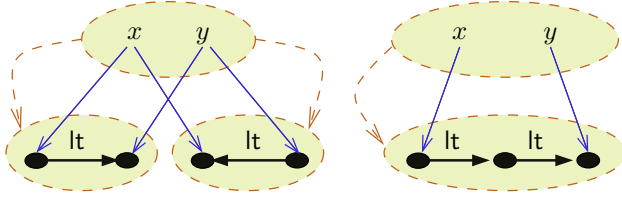## 3 Graph Predicates and Conditions

For *finite* $G \in$ Graph we define Pred$[G]$, Cond$[G]$ as the smallest sets such that

- $p \subseteq$ Cond$[G]$ with p finite implies $p \in$ Pred$[G]$;
- $\alpha \in$ Graph(G, H) and $p \in$ Pred$[H]$ implies $(\alpha, p) \in$ Cond$[G]$.

The elements of Pred$[G]$ are called *(graph) predicates on G* and those of Cond$[G]$ *(graph) conditions on G.* Graph predicates can be thought of as finitely branching trees of connected graph morphisms, of finite depth; a graph condition is a single branch of such a tree. For $c \in$ Cond$[G]$ we write α_c for the morphism component, $T_c = tgt(\alpha_c)$ for the target of α_c and p_c for the predicate component; hence $c = (\alpha_c, p_c)$. The depth of predicates and conditions, which is quite useful as a basis for inductive proofs, is defined by:

$$
\begin{aligned}
depth(p) &= \max \{ depth(c) \mid c \in p \} \\
depth(c) &= 1 + depth(p_c) \ .
\end{aligned}
$$

Fig. 1. Graph predicates for $\mathsf{lt}(x, y) \lor \mathsf{lt}(y, x)$ resp. $\exists z: \mathsf{lt}(x, z) \land \mathsf{lt}(z, y)$
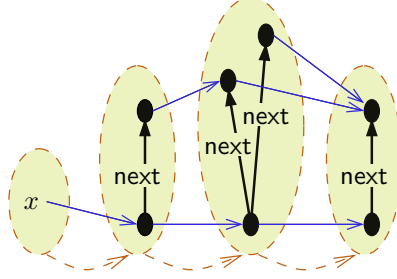

Fig. 2. Graph predicate for $\exists y: \mathsf{next}(x, y) \land \forall z: (\mathsf{next}(x, z) \Rightarrow z = y)$

It follows that the base case, $depth(p) = 0$, corresponds to $p = \emptyset$. Conditions have positive depth. We propose $\mathsf{Pred}[\langle X \rangle]$ as representations of FOL formulae over X. Note that in the introduction we discussed predicates consisting of a single condition only, and in the pictorial representation we omitted the source graph $\langle X \rangle$ (which anyway would be empty since the constraints discussed there are closed) and only displayed the structure from T_c onwards. Fig. 1 depicts two constraints with free variables accurately; Fig. 2 is another example, which shows multiple levels of conditions. The following defines satisfaction of a predicate $p \in \mathsf{Pred}[G]$, for arbitrary $\theta \in \mathsf{Graph}(G, H)$:

$$\theta \models p \ \text{ iff } \ \exists c \in p: \exists \mu: T_c \rightarrow H: \theta = \mu \circ \alpha_c, \mu \not\models p_c \ . \tag{1}$$

On the model side this generalizes $\models$ over FOL: here the source of θ can be an arbitrary graph, whereas there it was always discrete. An example is given in Fig. 3, which shows a model of the right-hand predicate of Fig. 1.
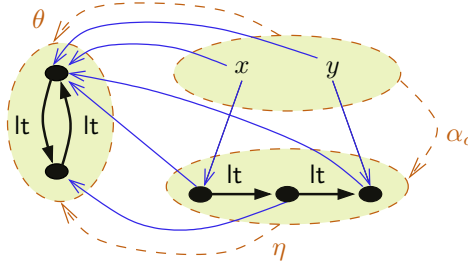
Fig. 3. Model satisfying the graph predicate for $\exists z: \mathsf{lt}(x, z) \land \mathsf{lt}(z, y)$

It follows that a predicate p should be seen as the *disjunction* of its conditions $c \in p$, and a condition c as the requirement that the structure encoded in T_c is present in the model, combined with the *negation* of p_c. This interpretation is formalized by recursively translating all $p \in \mathsf{Pred}[G]$ to formulae ϕ_p. For the translation, we assume that for all (sub-)conditions d occurring in the structure to be translated, the node sets of $src(\alpha_d)$ and $tgt(\alpha_d)$ are disjoint. (We show below that this assumption does not cause loss of generality, since we can always fulfill it by choosing isomorphic representatives of predicates and conditions.) Furthermore, we assume that for every node v occurring anywhere in the structure to be translated, there is a distinct associated variable x_v.

$$\phi_p = \bigvee \{\phi_c \mid c \in p\}$$
$$\phi_c = \exists \{x_v \mid v \in N_{T_c}\} : \bigwedge \{x_v = x_{\alpha_c(v)} \mid v \in N_G\} \wedge$$
$$\bigwedge \{a(x_v, x_w) \mid (v, a, w) \in E_{T_c}\} \wedge \neg \phi_{p_c} \ .$$

For every graph K occurring in p let $X_K = \{x_v \mid v \in N_K\}$ and let $\xi_K : \langle X_K \rangle \to K$ be given by $x_v \mapsto v$ for all $v \in N_K$. The following theorem is one of the main results of this paper.

Theorem 1. *Let $p \in \mathsf{Pred}[G]$ and $\theta \in \mathsf{Graph}(G, H)$; then $\theta \models p$ iff $\theta \circ \xi_G \models \phi_p$.*

Proof sketch. We prove the theorem together with an auxiliary result about conditions. First we extend the modeling relation to conditions, and we simplify the definition of $\models$ over Pred, as follows:

$$\theta \models c \text{ iff } \exists \mu : T_c \to H : \theta = \mu \circ \alpha_c, \mu \not\models p_c \tag{2}$$
$$\theta \models p \text{ iff } \exists c \in p : \theta \models c \ . \tag{3}$$

It follows that $\theta \models p$ iff $\theta \models c$ for some $c \in p$. The proof obligation is extended with:

If $c \in \mathsf{Cond}[G]$ then $\theta \models c$ iff $\theta \circ \xi_G \models \phi_c$.

The proof proceeds by mutual induction on these cases and on the depth of conditions. □

4   Formulae as Graph Predicates

We now provide the inverse translation from formulae into graph predicates. For this purpose we need some constructions over predicates. First, for $\mu \in \mathsf{Graph}(H, G)$, $p \in \mathsf{Pred}[G]$ and $c \in \mathsf{Cond}[G]$, we define

$$p \circ \mu = \{c \circ \mu \mid c \in p\}$$
$$c \circ \mu = (\alpha_c \circ \mu, p_c) \ .$$

Clearly, $p \circ \mu \in \mathsf{Pred}[H]$ and $c \circ \mu \in \mathsf{Cond}[H]$. This construction satisfies:

Proposition 2. *Let $p \in \mathsf{Pred}[H]$, $\mu \in \mathsf{Graph}(G, H)$ and $\theta \in \mathsf{Graph}(G, K)$; then $\theta \models p \circ \mu$ iff there is an $\eta \in \mathsf{Graph}(H, K)$ such that $\eta \models p$ and $\theta = \eta \circ \mu$.*

Proof: if. Assume $\eta \models p$. It follows that, for some $c \in p$, there is a $\lambda: T_c \to K$ such that $\eta = \lambda \circ \alpha_c$ and $\lambda \not\models p_c$. But then $\theta = \eta \circ \mu = \lambda \circ \alpha_c \circ \mu = \lambda \circ \alpha_{c \circ \mu}$; since $c \circ \mu \in p \circ \mu$, it follows that $\theta \models p \circ \mu$.

Only if. Assume $\theta \models p \circ \mu$. It follows that, for some $c \in p$, there is a $\lambda: T_c \to K$ such that $\theta = \lambda \circ \alpha_{c \circ \mu} = \lambda \circ \alpha_c \circ \mu$ and $\lambda \not\models p_{c \circ \mu}$ $(= p_c)$. But then $\lambda \circ \alpha_c \models p$, and hence $\eta = \lambda \circ \alpha_c$ fulfills the proof obligation. $\qquad\square$

In the sequel we make heavy use of pushouts in Graph; therefore we introduce some auxiliary notation. Given $\alpha: H \to K$ and $\mu: G \to H$, we write $\alpha \uparrow \mu$ ("the remainder of α after μ") for the morphism opposite α in the pushout diagram; hence $\alpha \uparrow \mu: G \to L$ and $\mu \uparrow \alpha: K \to L$ are such that (among other properties) $(\alpha \uparrow \mu) \circ \mu = (\mu \uparrow \alpha) \circ \alpha$. We extend this notation to predicates $p \in \mathsf{Pred}[G]$ and conditions $c \in \mathsf{Cond}[G]$ as follows:

$$p \uparrow \mu = \{c \uparrow \mu \mid c \in p\}$$
$$c \uparrow \mu = (\alpha_c \uparrow \mu, p_c \uparrow (\mu \uparrow \alpha_c)) \ .$$

It follows that $p \uparrow \mu \in \mathsf{Pred}[H]$ and $c \uparrow \mu \in \mathsf{Cond}[H]$. By taking pushouts, essentially we merge the additional structure specified by μ with the existing conditions, in the "least obtrusive" way. These constructions clearly yield predicates, resp. conditions again. The following correspondence plays an important role in the sequel.

Proposition 3. *Let $p \in \mathsf{Pred}[G]$, $\mu \in \mathsf{Graph}(G, H)$ and $\theta \in \mathsf{Graph}(H, K)$; then $\theta \circ \mu \models p$ iff $\theta \models p \uparrow \mu$.*

Proof sketch. The proof strategy is similar to that of Th. 1, by mutual induction on the depth of predicates and conditions, alternating between "case Pred" in the proposition, and "case Cond" reading "If $c \in \mathsf{Cond}[G]$, then $\theta \circ \mu \models c$ iff $\theta \models c \uparrow \mu$." $\qquad\square$

We now turn each $\mathsf{Pred}[G]$ and $\mathsf{Cond}[G]$ into a category. The arrows will essentially be proofs of implication. We define the hom-sets $\mathsf{Pred}[G](p, q)$ for $p, q \in \mathsf{Pred}[G]$ and $\mathsf{Cond}[G](c, d)$ for $c, d \in \mathsf{Cond}[G]$ as the families of smallest sets such that:

- $f: p \to q$ a function and $\gamma_c \in \mathsf{Cond}[G](c, f(c))$ a condition arrow for all $c \in p$ implies $(f, (\gamma_c)_{c \in p}) \in \mathsf{Pred}[G](p, q)$;
- $\mu: T_d \to T_c$ a function (in the reverse direction!) such that $\alpha_c = \mu \circ \alpha_d$ and $\pi \in \mathsf{Pred}[T_c](p_d \uparrow \mu, p_c)$ a compatible predicate arrow implies $(\mu, \pi) \in \mathsf{Cond}[G](c, d)$.

We let π range over sets of the form $\mathsf{Pred}[G](p, q)$, and use f_π and $\gamma_{\pi, c}$ for $c \in p$ to denote its components. Similarly γ ranges over sets of the form $\mathsf{Cond}[G](c, d)$, and μ_γ, π_γ denote its components. The following confirms the intuition that arrows between predicates are proofs of logical implication. The proof again goes by mutual induction (on the depth of p) of cases for Pred and Cond.

Proposition 4. *Let $\theta \in \mathsf{Graph}(G, H)$ and $p, q \in \mathsf{Pred}[G]$. If $\mathsf{Pred}[G](p, q)$ is non-empty then $\theta \models p$ implies $\theta \models q$.*

In preparation of the translation from FOL to graph predicates, we define the following operations over $c, d \in \mathsf{Cond}[G]$ and $p, q \in \mathsf{Pred}[G]$ (for arbitrary G):

$$c \times d = ((\alpha_c \uparrow \alpha_d) \circ \alpha_d, p_c \uparrow (\alpha_d \uparrow \alpha_c) \uplus p_d \uparrow (\alpha_c \uparrow \alpha_c)) \tag{4}$$

$$p + q = p \uplus q \tag{5}$$

$$p \times q = \{c \times d \mid c \in p, d \in q\} \tag{6}$$

$$!p = \{(id_G, p)\} \ . \tag{7}$$

In passing we note some facts about Pred and Cond.

Theorem 2. *For all $G \in$ Graph, $\mathsf{Cond}[G]$ is a category with products defined by (4) and initial object $(id_G, \emptyset)$; $\mathsf{Pred}[G]$ is a category with products defined by (6), coproducts defined by (5), initial object $\emptyset$ and terminal object $\{(id_G, \emptyset)\}$.*

Proof sketch. Concatenation in $\mathsf{Cond}[G]$ and $\mathsf{Pred}[G]$, for $\pi_i = (f_i, (\gamma_{i,c})_{c \in p_i}) \in \mathsf{Pred}[G](p_i, p_{i+1})$ and $\gamma_i = (\mu_i, \pi_i) \in \mathsf{Cond}[G](c_i, c_{i+1})$ $(i = 1, 2)$, is defined by

$$\pi_2 \circ \pi_1 = (f_2 \circ f_1, (\gamma_{2, f_1(c)} \circ \gamma_{1,c})_{c \in p_1}))$$
$$\gamma_2 \circ \gamma_1 = (\mu_1 \circ \mu_2, (\pi_2 \uparrow \mu_1) \circ \pi_1) \ .$$

where for $\lambda \in \mathsf{Graph}(G, H)$ and $\pi \in \mathsf{Pred}[G](p, q), \gamma \in \mathsf{Cond}[G](c, d)$, the remainders $\pi \uparrow \lambda \in \mathsf{Pred}[H](p \uparrow \lambda, q \uparrow \lambda)$ and $\gamma \uparrow \lambda \in \mathsf{Pred}[H](c \uparrow \lambda, d \uparrow \lambda)$ are defined by

$$\pi \uparrow \lambda = (\{(c \uparrow \lambda, f_\pi(c) \uparrow \lambda) \mid c \in p\}, (\gamma_c \uparrow \lambda)_{c \uparrow \lambda \in q \uparrow \lambda})$$
$$\gamma \uparrow \lambda = (\mu \uparrow \lambda, \pi \uparrow \lambda) \ .$$

Note that, in order for $\pi \uparrow \lambda$ to be well-defined, we need to assume distinct $c \uparrow \lambda$. Since we are free to choose these objects up to isomorphism, this assumption causes no loss of generality. The projections pr_c and pr_d for the product in $\mathsf{Cond}[G]$ are given by

$$pr_c = (\alpha_d \uparrow \alpha_c, id_{p_c \uparrow (\alpha_d \uparrow \alpha_c)}) \quad (\in \mathsf{Cond}[G](c \times d, c))$$
$$pr_d = (\alpha_c \uparrow \alpha_d, id_{p_d \uparrow (\alpha_c \uparrow \alpha_d)}) \quad (\in \mathsf{Cond}[G](c \times d, d)) \ .$$

$\square$

The following proposition affirms that the operations defined above are appropriate for modeling FOL connectives. This partially follows from the characterization (in Th. 2) of $+$ and $\times$ as coproduct and product in Pred, plus Prop. 4 stating that arrows in Pred induce logical implication.

Proposition 5. *Let $\theta \in$ Graph(G, H) and $p, q \in$ Pred$[G]$.*

1. *$\theta \models p + q$ if and only if $\theta \models p$ or $\theta \models q$.*
2. *$\theta \models p \times q$ if and only if $\theta \models p$ and $\theta \models q$.*
3. *$\theta \models !p$ if and only if $\theta \not\models p$.*

The final elements we need for the translation from FOL to graph predicates are representations for the base formulae, $x = y$ and $\mathsf{a}(x, y)$. These are given through graph morphisms $\alpha_{x=y}$ and $\alpha_{\mathsf{a}(x,y)}$, given pictorially in Fig. 4. The following table defines a function yielding for every $\phi \in$ FOL and finite $X \supseteq fv(\phi)$ a graph predicate $\llbracket \phi \rrbracket_X \in \mathsf{Graph}[\langle X \rangle]$.
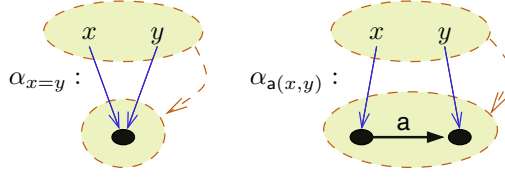
Fig. 4. Graph morphisms for the base formulae $x = y$, resp. $\mathsf{a}(x,y)$

$$[\![x = y]\!]_X = \{(\alpha_{x=y}, \emptyset)\}{\uparrow} emb[\{x,y\}, X]$$
$$[\![\mathsf{a}(x,y)]\!]_X = \{(\alpha_{\mathsf{a}(x,y)}, \emptyset)\}{\uparrow} emb[\{x,y\}, X]$$
$$[\![\neg\phi]\!]_X = \,![\![\phi]\!]_X$$
$$[\![\bigvee \Phi]\!]_X = \sum_{\phi \in \Phi}[\![\phi]\!]_X$$
$$[\![\bigwedge \Phi]\!]_X = \prod_{\phi \in \Phi}[\![\phi]\!]_X$$
$$[\![\exists Y.\phi]\!]_X = [\![\phi]\!]_{X \cup Y} \circ emb[X, X \cup Y] \ .$$

The following is the second half of the main correspondence result of this paper.

Theorem 3. *Let $\phi \in$ FOL and $\theta \in \mathsf{Graph}(\langle X\rangle, G)$ with $X \supseteq fv(\phi)$; then $\theta \models \phi$ iff $\theta \models [\![\phi]\!]_X$.*

Proof. By induction on the structure of ϕ. For the base formulae the result is immediate. For negation, disjunction and conjunction the result follows from Prop. 5, and for the existential from Prop. 2. □

It should be clear that there is a direct connection between the depth of graph predicates and the level of nesting of negation in the corresponding formula. We will make this connection precise. We use paths through the syntax tree of the formulae to isolate the relevant fragments of FOL. In the following theorem, a string of $\exists$ and $\neg$ indicates the set of all formulae for which, if we follow their syntax trees from the root to the leafs and ignore all operators except $\exists$ and $\neg$, all the resulting paths are prefixes of that string.

Theorem 4. *Let $n \in$ Nat; then the set of predicates p with $depth(p) \leq n$ is equivalent to the FOL-fragment $\exists(\neg\exists)^n$.*

Note that we could have formulated the same result while omitting $\exists$; we have included it to stress that $\forall$ is only allowed in its dual form, $\neg\exists\neg$.

Proof. Immediate from the two conversion mappings, ϕ_p for $p \in$ Pred and $[\![\phi]\!]$ for $\phi \in$ FOL; see Th. 1 and Th. 3. □

This result has consequences in the theory of algebraic graph transformations. The key insight is that the application conditions of [10] are exactly graph predicates of depth 0 (the positive conditions) and 1 (the negative conditions, or NACs) — where it should be noted that application conditions are closed formulae, so the corresponding graph predicates are in Pred$[\langle\emptyset\rangle]$; and indeed the presentation in [10] omits the base graph.

Corollary 1. *The application conditions proposed in [10] are equivalent to the FOL-fragment $\exists\neg\exists$.*

For instance, a useful property that can yet *not* be expressed through NACs is the uniqueness of nodes: the property $\exists y\colon \mathsf{next}(x,y) \wedge \forall z\colon (\mathsf{next}(x,z) \Rightarrow z = y)$, expressed in Fig. 2 by a graph predicate of depth 2, is outside the fragment $\exists\neg\exists$. More generally, as noted in the conclusion of [10], NACs cannot impose cardinality constraints, which in fact all have roughly the form $\exists X\colon \left(\bigwedge_{x,y\in X} x \neq y\right) \wedge \forall z\colon \bigvee_{x\in X} z = x$, and hence are in $\exists\neg\exists\neg$.[3]

5   Graph Predicates as Graphs

The way we defined them, graph predicates are highly structured objects. We now show that much of this structure can be done away with: there is an equivalent representation of graph conditions as sets of simple, flat graphs, in which the nesting structure is transferred to the *edge labels*. Graph predicates thus correspond to *sets* of such graphs. In this section we define the conversions back and forth. All proofs, being rather technical and uninteresting, are omitted. In this section, Lab is as follows:

$$\mathsf{Lab} = \mathsf{Rel} \cup (\mathsf{Rel}_= \times \mathsf{Nat}) \ .$$

- Rel is the set of relation symbols, as before. Plain relation symbols (i.e., without depth indicators, see below) are called *base labels*; they correspond to the base graph G of a predicate $p \in \mathsf{Pred}[G]$.
- $\mathsf{Rel}_= \times \mathsf{Nat}$, where $\mathsf{Rel}_= = \mathsf{Rel} \cup \{=\}$, consists of pairs of a relation symbol (this time including equality $=$, see below), together with a natural number indicating the *depth* of the edge. For Rel this will be the depth in p at which the edge is introduced; for $=$ it will be the depth at which the nodes are introduced or equated.

We use b to range over $\mathsf{Rel}_=$ and $^i\mathsf{b}$ as shorthand for (b, i); we use ℓ to range over Lab. Furthermore, we use $\perp$ to denote the *base depth* and regard it as an element smaller than any $i \in \mathsf{Nat}$, and such that $i - 1 = \perp$ if $i \in \{\perp, 0\}$. We use δ, ϵ to range over $\mathsf{Nat} \cup \{\perp\}$ and sometimes use $^\perp\mathsf{a}$ as equivalent notation for a ($\in \mathsf{Rel}$). We define the *depth* of nodes and edges as

- $depth(v) = i$ if $(v, \overset{i}{=}, v) \in E$, and $\perp$ otherwise;
- $depth(e) = i$ if $e = (v, {}^i\mathsf{a}, w)$.

(Node depth is well-defined due to the first well-formedness constraint below.) We call $a \in N \cup E$ *base* if $depth(a) = \perp$. In the remainder of this section we use RelGraph to denote the set of graphs over Rel used in the previous sections, and CondGraph to denote the set of *condition graphs*, which are graphs over Lab that satisfy the following well-formedness constraints, in addition to those already mentioned above:

---

[3] This is no longer true when matchings are required to be injective; see Sect. 6 for a conjecture about the increased expressiveness of that setting. Also, as remarked in the introduction, the double-pushout approach imposes further application conditions to ensure the existence of pushout complements, which we ignore here.
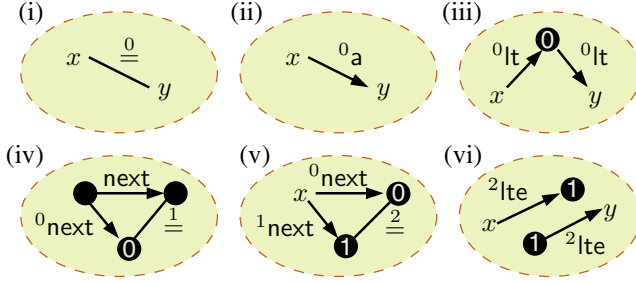
Fig. 5. Example condition graphs.

1. For any $v, w \in N$ and $\mathsf{b} \in \mathsf{Rel}_=$ there is at most one δ such that $(v, {}^\delta\mathsf{b}, w) \in E$.
2. If $(v, {}^\delta\mathsf{b}, w) \in E$ then $depth(v), depth(w) \leq \delta$.
3. If $(v, \overset{i}{=}, w) \in E$ then $(w, \overset{i}{=}, v) \in E$.

Note that $\mathsf{RelGraph} \subseteq \mathsf{CondGraph}$. Fig. 5 contains some example condition graphs. We indicate node depths by inscribing the depth inside nodes, and edge depths by appending the depth to the label. $\overset{i}{=}$-labeled edges, which are always bidirectional due to well-formedness condition 3, are drawn undirected. Graphs (i) and (ii) are flattenings of the morphisms $\alpha_{x=y}$ and $\alpha_{\mathsf{a}(x,y)}$ displayed in Fig. 4. Graph (iii) is the flattening of the right hand condition of Fig. 1; (iv) is the condition of Fig. 2 without the base level and (v) the complete condition. Graph (vi) represents the (right hand, connected) condition of Fig. 7 below.

Any graph morphism $\alpha \colon G \to C$ can be flattened to a condition graph, $flat(\alpha)$, by enriching G with the structure provided by α while keeping it distinguishable from the structure already present in G, so that α can be fully reconstructed (up to isomorphism of C). There are essentially three kinds of additional structure: fresh nodes of C, fresh edges of C, and node mergings, i.e., nodes on which α is non-injective. Node mergings and fresh nodes will be indicated using $\overset{0}{=}$-labeled edges, and fresh edges by a (non-base) depth indication. In general we allow $G \in \mathsf{RelGraph}$ and $C \in \mathsf{CondGraph}$. W.l.o.g. assume $N_G \cap N_C = \emptyset$; then $flat(\alpha) = (N, E) \in \mathsf{CondGraph}$ such that

$$N = N_G \cup (N_C \setminus \alpha(N_G))$$
$$E = E_G \cup \{(u, {}^0\mathsf{a}, v) \mid (u, \mathsf{a}, v) \notin E_G, (\bar{\alpha}(u), \mathsf{a}, \bar{\alpha}(v)) \in E_C\}$$
$$\cup \{(u, \overset{0}{=}, v) \mid u \neq_G v, \bar{\alpha}(u) = \bar{\alpha}(v)\}$$
$$\cup \{(u, {}^{i+1}\mathsf{b}, v) \mid (\bar{\alpha}(v), {}^i\mathsf{b}, \bar{\alpha}(v)) \in E_C\} \ .$$

Here $\bar{\alpha} = \alpha \cup id_{N \setminus N_G}$ extends α with identity mappings for the fresh nodes, and $=_G$ is the identity relation over N_G. Note that we do not only add depth indicators to the additional structural elements, but we also increment the depth indicators already present in C, so as to keep them distinct. For instance, note that, as expected, graphs (i) and (ii) in Fig. 5 indeed equal $flat(\alpha_{x=y})$ and $flat(\alpha_{\mathsf{a}(x,y)})$ (see Fig. 4).

For the inverse construction, we need to resurrect the original source and target graphs from the flattened morphism; or more generally, we construct morphisms from conditional graphs. In principle, the source graph is the sub-graph with base depth,

whereas the target graph is the the entire condition graph, where, however, the depths are decremented and the nodes connected by $\overset{0}{=}$-edges are collected into single nodes.

In the following, given a binary relation ρ over a set A, for any $a \in A$ we define $[a]_\rho$ as the smallest set such that (i) $a \in [a]_\rho$, and (ii) if $b \in [a]_\rho$ and $\rho(b,c)$ or $\rho(c,b)$ then $c \in [a]_\rho$. Likewise, $A/\rho = \{[a]_\rho \mid a \in A\}$ is the partitioning of A according to ρ.

Let $G \in \mathsf{RelGraph}$, $C \in \mathsf{CondGraph}$ and $\mu = (G, C, f)$. We define $C|_\perp$ as the base part of C, and C^- as C considered "one level up", i.e., with all depth indicators decremented and all $\overset{0}{=}$-related nodes collected (considering $v \overset{0}{=} w$ iff $(v, \overset{0}{=}, w) \in E$). Using these, we construct $\phi_C \colon C|_\perp \to C^-$ mapping each $v \in N_C|_\perp$ to $[v]_{\overset{0}{=}}$.

$$C|_\perp = (N_C|_\perp, E_C|_\perp) \text{ where } A|_\perp = \{a \in A \mid depth(a) = \perp\} \tag{8}$$

$$C^- = (N/\overset{0}{=}, \{([u]_{\overset{0}{=}}, {}^{\delta-1}\mathsf{b}, [v]_{\overset{0}{=}}) \mid (u, {}^\delta\mathsf{b}, v) \in E_C \wedge (\mathsf{b} \in \mathsf{Rel} \vee \delta > 0)\}) \tag{9}$$

$$\mu|_\perp = (G, C|_\perp, f) \tag{10}$$

$$\phi_C = (C|_\perp, C^-, \{(v, [v]_{\overset{0}{=}}) \mid v \in N_C|_\perp\}) \ . \tag{11}$$

Resurrecting a morphism is left inverse to flattening, but not for all condition graphs right inverse. The latter is due to the fact that we have not bothered to give an exact characterization of those condition graphs that may be constructed by flattening. Such a characterization would be cumbersome and not add much to the current paper: the main purpose here is to show that graph conditions may be flattened without loss of information.

Proposition 6. *Let $G \in \mathsf{RelGraph}$ and $C \in \mathsf{CondGraph}$ with $\alpha \colon G \to C$.*

1. *There exists an isomorphism μ such that $\phi_{flat(\alpha)} = \alpha \circ \mu$.*
2. *There exists an epimorphism $\mu \colon C \to flat(\phi_C)$.*

We now extend these principles to graph conditions c, which are essentially nested morphisms. Here the depth indicators really come into play. The construction proceeds by flattening the sub-conditions in p_c, taking the union of the resulting graph as an extended target graph for α_c and then flattening (the extended) α_c.

$$flat(c) = flat(\beta) \text{ where } \beta = emb[T_c, \bigcup_{d \in p_c} flat(d)] \circ \alpha_c \tag{12}$$

where $emb[G, H] \in \mathsf{Graph}(G, H)$ is given by (G, H, id_{N_G}) if $N_G \subseteq N_H$, $E_G \subseteq E_H$.

For the inverse construction, we need to reconstruct the $d \in p_c$ from $flat(c)$. For this purpose, we use the *connectedness* of $flat(c)$. A fragment of a condition graph will be taken as part of the same sub-condition if it is connected at depth > 0. For instance, graph (v) in Fig. 5 has one connected sub-condition, whereas graph (vi) has two.

The required notion of connectedness can be captured through the decomposition of morphisms into *primes*, as follows. For $\lambda \in \mathsf{Graph}(G, H)$ and $\mu \in \mathsf{Graph}(G, K)$ we define $\lambda +^G \mu = (\lambda \uparrow \mu) \circ \mu$; the superscript G stands for the source graph of the morphisms considered — more formally, this is the coproduct operation in a category of morphisms with source G (the slice category of Graph under G). Connectedness, in the above sense, is related to the ability to decompose a morphism into summands. We call μ *prime* if it has no non-trivial decomposition under $+^G$; that is, if $\mu = \sum^G M$ for a set of morphisms M (where $\sum^G \emptyset = id_G$) implies that M contains some isomorphic representative of μ. The following characterizes prime morphisms.
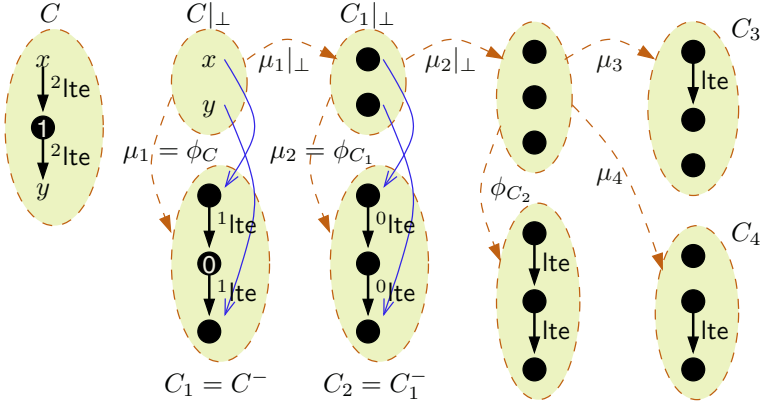
Fig. 6. Steps in the construction of a graph condition

Proposition 7. $\mu \in \mathsf{Graph}(G, H)$ *is prime iff one of the following holds:*

1. *μ is an epimorphism, and non-injective on exactly one pair of nodes;*
2. *μ is a monomorphism, and the set A of nodes and edges in H that do not occur as images of μ is connected by $\leftrightarrow \subseteq A \times A$, defined as the least relation such that $(v, \ell, w) \leftrightarrow v$ and $(v, \ell, w) \leftrightarrow w$ whenever $(v, \ell, w), v, w \in A$.*

The key property in the use of prime morphisms, stated in the following proposition, is that every morphism $\mu \colon G \to H$ can be decomposed into a finite number of them.

Proposition 8. *For all $\mu \in \mathsf{Graph}(G, H)$, there is a finite set of prime morphisms P, such that $\mu = \alpha \circ \sum^G P$ for some isomorphism α.*

Note that the decomposition P is not unique, even up to isomorphism; however, if μ is an isomorphism then $P = \emptyset$ is the only possibility. For the developments in this section it does not matter which decomposition we take; rather, we assume that $primes(\mu)$ is *some* (fixed) prime decomposition. Let $G \in \mathsf{RelGraph}$, $C \in \mathsf{CondGraph}$ and $\mu \colon G \to C$.

$$cond(\mu) = (\mu|_\perp, \{cond(\eta) \mid \eta \in primes(\phi_{tgt(\mu)})\}) \tag{13}$$
$$cond(C) = cond(\phi_C) . \tag{14}$$

Thus, *cond* constructs a graph condition from a morphism by turning its target graph into a new morphism, and recursively calling itself on its prime decomposition. This terminates because in $\eta \in primes(\phi_D)$ with $D = tgt(\mu)$, all depth indicators of $tgt(\eta)$ have been decreased w.r.t. D; and if D is base then ϕ_D is an iso, hence $primes(\phi_D) = \emptyset$. For example, Fig. 6 shows several stages of constructing $cond(C)$, with C the graph on the left hand side. ϕ_C and ϕ_{C_1} are themselves prime, but $primes(\phi_{C_2}) = \{\mu_3, \mu_4\}$. From the figure we can see $cond(C) = (\mu_1|_\perp, \{(\mu_2|_\perp, \{(\mu_3|_\perp, \emptyset), (\mu_4|_\perp, \emptyset)\})\})$.

This construction gives us half of the desired correspondence (compare Prop. 6.2):

Proposition 9. *All $C \in \mathsf{CondGraph}$ have an epimorphism $\mu \colon C \to flat(cond(C))$.*
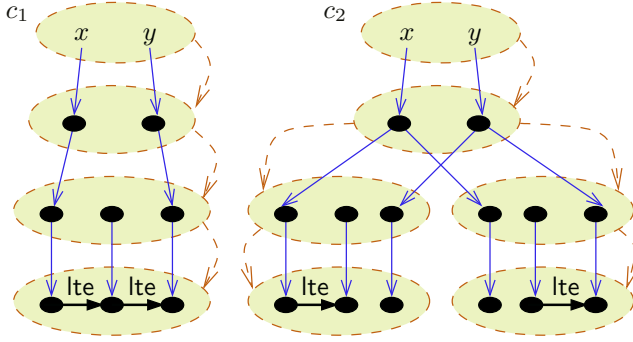
Fig. 7. An unconnected graph condition and its connected normal form

Still, this does not yet solve the problem, since graph conditions do not generally have the connectedness required to reconstruct them from their flattenings. For instance, $flat(c_1)$ for c_1 as in Fig. 7 yields C of Fig. 6, but $cond(C)$ is not isomorphic to c_1, and indeed the two are also inequivalent as properties over graphs. On the other hand, c_2 in Fig. 7 is equivalent to c_1 in this sense; $flat(c_2)$ yields graph (vi) of Fig. 5, from which $cond$ does construct a condition isomorphic to c_2.

To formulate the required connectedness property, we *enrich* the target graph T_c of conditions c with information about the connections made deeper in the tree of morphisms underlying c. For arbitrary $p \in \mathsf{Pred}[G]$ and $c \in \mathsf{Cond}[G]$ we define:

$$G_p^{\leftrightarrow} = (N_G, E_G \cup \{(v, \leftrightarrow, w) \mid \exists c \in p : \alpha_c(v) \text{ is connected to } \alpha_c(w) \text{ in } (T_c)_{p_c}^{\leftrightarrow}\})$$
$$\alpha_c^{\leftrightarrow} = (G, (T_c)_{p_c}^{\leftrightarrow}, f_{\alpha_c}) \ .$$

This brings us to the following definition of connectedness.

Definition 5.1. *Graph condition* $c \in \mathsf{Cond}[G]$ *is called* connected *if for all* $d \in p_c$, $\alpha_d^{\leftrightarrow}$ *is prime and* d *is connected.*

The following proposition lists the important facts about connected graph conditions: the graph conditions constructed by $cond$ (14) are always connected, connected graph conditions can be flattened without loss of information (compare Prop. 6.1), and for every graph condition there is an equivalence connected one.

Proposition 10. *Let* $c \in \mathsf{Cond}[G]$.

1. *If* $c = cond(C)$ *for some* $C \in \mathsf{CondGraph}$, *then* c *is connected.*
2. *If* c *is connected, then there is an isomorphism* $\gamma \colon c \to cond(flat(c))$.
3. *There is a connected* $\bar{c} \in \mathsf{Cond}[G]$ *such that* $\theta \models c$ *if and only if* $\theta \models \bar{c}$.

This brings us to the main result of this section, which states that every graph condition can be flattened to a condition graph expressing the same property. In order to represent a graph predicate $p \in \mathsf{Pred}[G]$, we use the *set* of condition graphs $\{flat(c) \mid c \in p\}$.

Theorem 5. *Let* $c \in \mathsf{Cond}[G]$; *then there is a* $C \in \mathsf{CondGraph}$ *such that* $\theta \models cond(C)$ *if and only if* $\theta \models c$.

6   Conclusions

We have presented an equivalent representation of first-order logic, as a recursively nested set of graph morphisms. We have defined compositional translations back and forth, and given an expressiveness result relating the recursive depth of our graph predicates to the corresponding fragment of FOL.

Subsequently, we have shown how the nested graph predicates can be translated, without loss of information, to flat graphs. We see as the main advantage of this that we can now use graph transformations to transform predicate graphs. This points the way to a potential connection between graph transformation and predicate transformation.

Graph constraints and conditional application conditions. At several points during the paper we already mentioned the work on (negative) application conditions within the theory of algebraic graph transformation, originally by [6] and later worked out in more detail in [10]. This paper is the result of an attempt to generalize their work.

Other related work in the context of graph transformations may be found in the *conditional application conditions* of [11] and in the *graph constraints* as proposed in, e.g., [12] and implemented in the AGG tool (cf. [9]). In fact it is not difficult to see, using the results of this paper, that conditional application conditions are expressively equivalent to the ∃¬∃¬∃-fragment of FOL, and graph constraints to the ¬∃¬∃-fragment.

We conjecture that the requirement that all matches be injective increases the expressive power precisely by allowing *inequality* (but no other forms of negation) to occur in the context of the inner ∃.

Existential graphs. A large body of related work within (mainly) *artificial intelligence* centers around Peirce's *existential graphs* [14,15] and Sowa's more elaborate *conceptual graphs* [17]. The former were introduced as pragmatic philosophical models for reasoning over a century ago; the latter are primarily intended as models for knowledge representation. There are obvious similarities between those models and ours, especially concerning the use of nesting to represent negation and existential quantification. On the other hand, the thrust of the developments in existential and conceptual graphs is quite different from ours, making a detailed comparison difficult. New in the current paper seems to be the use of edge labels to encode the nesting structure; throughout the work on existential and conceptual graphs we have only seen this represented using so-called *cuts*, which are essentially sub-graphs explicitly describing the hierarchical structure. For us, the advantage of our approach is that our models can be treated as simple graphs, and as such submitted to existing graph transformation approaches.

More or less successful approaches to define a connection between existential, resp. conceptual, graphs and FOL can be found in [2,18,5]. In particular, in [18,5]) a complete theory of FOL is developed on the basis of the graph representations.

Variations. We have chosen a particular category of graphs and a particular encoding of FOL that turn out to work well together. However, it is interesting also to consider variations in either choice. For instance, our graphs do not have parallel edges, and our encoding does not allow reasoning about edge labels. It is likely that similar results can be obtained in an extended setting by moving to graph logics as in [1,4].

As an example of the use of such extensions, consider the so-called *dangling edge condition* that partially governs the applicability of a double-pushout rule (see also Footnote 1). This condition (under certain circumstances) forbids the existence of any edges outside an explicitly given set. In the setting of this paper, there is no uniform way to express such a constraint, since it requires the ability to refer to edges explicitly while abstracting from their labels.

Open issues. We list some questions raised by the work reported here.

- A direct semantics for condition graphs. The flat graph representations of Sect. 5 are shown to be equivalent by a translation back and forth to the nested graph predicate structures. Currently we do not have a modeling relation directly over condition graphs.
- The connection to predicate transformations. Traditional approaches to predicate transformation have to go to impressive lengths to represent pointer structures (e.g., [13]), whereas on the other hand graphs are especially suitable for this. Using the condition graph representation of predicates presented here, one can use graph transformations to construct or transform predicates.
- The extension of existing theory on graph transformation systems to support full graph predicates instead of (conditional) application conditions; for instance, rule independency in the context of negative application conditions developed in [10], and the translation of postconditions to application conditions in [11,12,7].

References

1. M. Bauderon and B. Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20(2–3):83–127, 1987.
2. M. Chein, M.-L. Mugnier, and G. Simonet. Nested graphs: A graph-based knowledge representation model with FOL semantics. In Cohn, Schubert, and Shapiro, eds., *Principles of Knowledge Representation and Reasoning*, pp. 524–535. Morgan Kaufmann, 1998.
3. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation, part I: Basic concepts and double pushout approach. In Rozenberg [16], chapter 3, pp. 163–246.
4. B. Courcelle. The monadic second-order logic of graphs XI: Hierarchical decompositions of connected graphs. *Theoretical Comput. Sci.*, 224(1–2):35–58, 1999.
5. F. Dau. *The Logic System of Concept Graphs with Negation*, vol. 2892 of *LNCS*. Springer, 2003.
6. H. Ehrig and A. Habel. Graph grammars with application conditions. In Rozenberg and Salomaa, eds., *The Book of L*, pp. 87–100. Springer, 1986.
7. H. Ehrig, K. Ehrig, A. Habel, and K.-H. Pennemann. Constraints and application conditions: From graphs to high-level structures. In *International Conference on Graph Transformations*, LNCS. Springer, 2004. This volume.
8. H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic approaches to graph transformation, part II: Single pushout approach and comparison with double pushout approach. In G. Rozenberg [16], pp. 247–312.
9. C. Ermel, M. Rudolf, and G. Taentzer. The AGG approach: language and environment. In Ehrig et al., eds., *Handbook of Graph Grammars and Computing by Graph Transformation*, vol. II: Applications, Languages and Tools. World Scientific, Singapore, 1999.

10. A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 26(3/4):287–313, 1996.

11. R. Heckel and A. Wagner. Ensuring consistency of conditional graph grammars — a constructive approach. *ENTCS*, 2, 1995.

12. M. Koch, L. V. Mancini, and F. Parisi-Presicce. Conflict detection and resolution in access control policy specifications. In Nielsen and Engberg, eds., *Foundations of Software Science and Computation Structures*, vol. 2303 of *LNCS*, pp. 223–238. Springer, 2002.

13. C. Pierik and F. S. de Boer. A syntax-directed hoare logic for object-oriented programming concepts. In Najm, Nestmann, and Stevens, eds., *Formal Methods for Open Object-based Distributed Systems*, vol. 2884 of *LNCS*, pp. 64–78. Springer, 2003.

14. D. D. Roberts. *The Existential Graphs of Charles S. Peirce*. Mouton and Co., 1973.

15. D. D. Roberts. The existential graphs. *Computers and Mathematics with Applications*, 6:639–663, 1992.

16. G. Rozenberg, ed. *Handbook of Graph Grammars and Computing by Graph Transformation*, vol. I: Foundations. World Scientific, Singapore, 1997.

17. J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.

18. M. Wermelinger. Conceptual graphs and first-order logic. In *International Confence on Conceptual Structures*, vol. 954 of *LNAI*, pp. 323–337. Springer, 1995.