Institutions for Navigational Logics for Graphical Structures[☆]

Fernando Orejas, Elvira Pino

Universitat Politècnica de Catalunya, Barcelona, Spain

Marisa Navarro

Universidad del País Vasco (UPV/EHU), San Sebastián, Spain

Leen Lambers

Hasso Plattner Institut, University of Potsdam, Germany

Abstract

We show that a Navigational Logic, i.e., a logic to express properties about graphs and about paths in graphs is a semi-exact institution. In this way, we can use a number of operations to structure and modularize our specifications. Moreover, using the properties of our institution, we also show how to structure single formulas, which in our formalism could be quite complex.

Keywords: Institutions, Graph Logics, Navigational Logics

1. Introduction

The extensive use of graphs in all areas of Computer Science is the reason for the relevance of being able to express graph properties and to reason about them. In particular, we are interested in the area of software modeling where, in the context of graphical modeling formalisms, like the UML, graph properties may be used to express constraints for a given model, and we are also interested in the area of graph databases, where graph properties may be used not only to express

[☆]This work has been partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grant COMMAS (ref. TIN2013-46181-C2-1-R, TIN2013-46181-C2-2-R) and from the Basque Project GIU15/30, and grant UFI11/45.

database constraints, but where a graph logic may be used as a basis to define a query language.

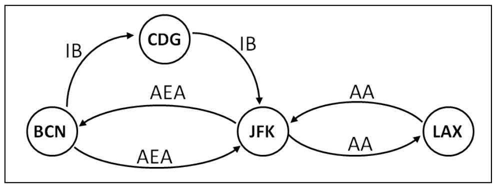
We may approach the problem of defining a graph logic in two different ways. On the one hand, we may just use a standard logic, after extending it with some graph concepts. For instance, Courcelle (e.g., [4]), studied a graph logic defined in terms of standard first-order (or monadic second-order) logic including some specific graph predicates. Similarly, in the area of graph databases, where foundational work has concentrated mainly on studying the expressivity or the complexity of classes of graph queries and other kind of related problems, they have studied extensions of first-order logic with classes of navigational path queries (see, e.g. [3, 1]). On the other hand, we may define a specific logic where formulas include graphs (and graph morphisms) as first-class citizens, like in the *logic of nested graph conditions* (LNGC), introduced by Habel and Pennemann [6], which was proven to be equivalent to the first-order logic of graphs of Courcelle. A main advantage of LNGC is that it is generic, since it can be used for any category of graphical structures, provided that this category enjoys certain properties. If this is not the case we need a different encoding for each class of graphs. In addition, from a practical point of view, Pennemann [11] showed that a specialized prover for their logic outperformed some standard provers when applied to graph formulas using Courcelle's logic.

A main problem of (first-order) graph logics is that it is not possible to express relevant properties like "there is a path from node n to n'", because they are not first-order. As a consequence, there have been a number of proposals that try to overcome this limitation by extending existing logics, like [7, 12, 8]. In particular, in [8] we extended the LNGC, allowing us to state properties about paths in graphs and to reason about them in a generic way (i.e. for arbitrary categories of graphical structures). Since this new logic allows one to describe properties of paths in graphical structures, we have called it a *navigational logic*.

Institutions were introduced in [5] to define the semantics of the Clear specification language, independently of any specification formalism. Showing that a given formalism is an institution allows us to use a number of constructions to structure and modularize our specifications [13]. For this reason, in this paper we show that a given navigational logic is a semi-exact institution. Moreover, using the properties of our institution, we also show how to structure single formulas, which in our formalism could be quite complex. For simplicity, in this paper we work with the specific category of labeled graphs, but the results can be generalized to arbitrary categories of graphical structures, following the lines of [8].

2. Navigational Logics for Graphical Structures: Introductory Examples

The idea of our logics is that basic properties state if a given pattern is present or not in a graph. In our case, a pattern is like a graph but, in addition to normal edges, we may have other types of edges (depicted here as double arrows) representing paths between two given nodes. For example, let us suppose that we want to express some properties about graphs that represent networks of airports, like the one in Fig. 1, where each node is labeled with the name of an airport and each edge represents a direct flight between the source and target airports and is labeled with the name of the airline running the flight. Moreover, we assume that paths are labeled with regular expressions over the edge labels. In this context, in Fig. 2, we depict two patterns, where the first one represents a connection from BCN to LAX consisting of a sequence of IB flights followed by an AA flight, and the second one a direct flight from BCN to CDG followed by a connection from CDG to LAX consisting of an IB flight followed by a sequence of AA flights.



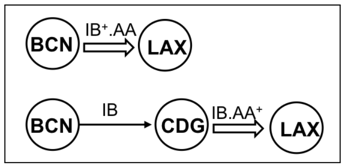


Figure 1: A graph of connected airports Figure 2: Two connection patterns

Formulas in our logics are built over patterns (and pattern morphisms) using quantifiers and the standard logical connectives. For example, in Fig. 3 we depict (in a pseudo-formal notation) two formulas to provide some intuition. The first one states that there must exist an IB flight from BCN to CDG followed by a sequence of AA flights leading to LAX, or there must exist a sequence of IB flights leading to JFK from BCN, followed by an AA flight to LAX. In the second one the long arrow denotes a morphism (the obvious inclusion morphism, in this case) between the pattern on the left (quantified universally) and the pattern on the right (quantified existentially), meaning that the target pattern is an extension of the source pattern. In particular, this formula states that for every connection between any two airports, there is a backward connection between them.[1]

[1]Here x and y represent any airline, i.e. $x, y = IB|AEA|AA|\ldots$.

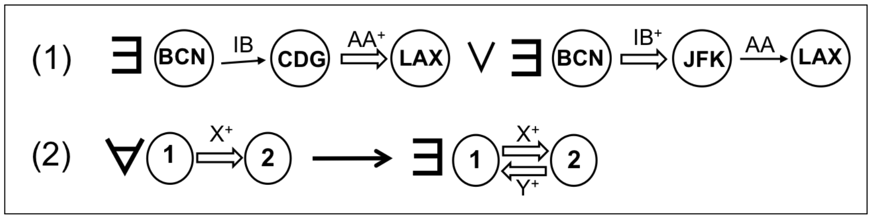


Figure 3: Properties on airports networks

3. A Navigational Logic

In this section we introduce formally our navigational logic for the case where the given graphs are labeled graphs. However, as shown in [8], these constructions can be generalized to arbitrary categories of graphs or graphical structures, where paths can be labeled not only by regular expressions, but by arbitrary language expressions. We start defining patterns:

Definition 1 (Labeled Graph Patterns and Paths). A *graph pattern* P is a pair $P = (G_P, \Rightarrow_P)$ such that G_P is a labeled graph and $\Rightarrow_P$ is a relation specifying paths in P, i.e., a set of *path expressions* of the form $\langle n, \alpha, n' \rangle$ where $n, n' \in Nodes_{G_P}$ and α is a regular expression over an alphabet Σ of edge labels, such that its associated language $\mathcal{L}(\alpha)$ is not empty. Then, a *path* specified by a path expression $\langle n, \alpha, n' \rangle$, is any triple $\langle n, s, n' \rangle$ such that $s \in \mathcal{L}(\alpha)$.

A pattern morphism $f : P_1 \to P_2$, is a graph morphism $f : G_{P_1} \to G_{P_2}$ such that $\langle n, \alpha, n' \rangle \in \Rightarrow_{P_1}$ implies $\langle f(n), \alpha', f(n') \rangle \in (\to_{P_2} \cup \Rightarrow_{P_2})^*$, for some α' with $\mathcal{L}(\alpha') \subseteq \mathcal{L}(\alpha)$, where $\to_P$ is the least relation satisfying that $\langle n, l, n' \rangle \in \to_P$, if there is an edge $e = n \xrightarrow{l} n'$ in G_P.

The class of patterns and pattern morphisms, form the category Patterns. A graph G can be considered as a kind of pattern where the relation $\Rightarrow_G$ exactly specifies the paths defined by edges in G. Therefore, we can assume that the category of graphs, Graphs, is the full subcategory of Patterns whose objects are of the form $(G, \to_G^*)$.

It may be easily proved that Patterns has colimits. In particular, colimits in Patterns can be built like colimits for a category of graphs with two kinds of edges (normal edges and paths).

As said above, the formulas used as examples in Sect. 2 were depicted in a pseudo-formal format. Now, we define precisely their syntax, using the nested notation defined in [6], and their semantics in terms of morphisms.

Definition 2 (Conditions and Satisfaction). Given a finite pattern P (i.e. G_P and $\Rightarrow_P$ are finite), a *condition with context* P, denoted c_P, is defined inductively as follows:

- $c_P = \mathtt{true}$.
- $c_P = \exists(a : P \to Q, c_Q)$ if Q is a finite pattern and c_Q is a condition with context Q.
- $c_P = \neg c'_P$.
- $c_P = c'_P \wedge c''_P$ if c'_P and c''_P are conditions with context P.

Given G in $\underline{\mathtt{Graphs}}$, and a morphism $f : P \to G \in Morph(\underline{\mathtt{Patterns}})$, we inductively define when f *satisfies* a condition c_P, denoted $f \models c_P$:

$$\begin{tikzcd}
P \arrow[rr, "a"] \arrow[dr, "f"'] & & Q \lhd c_Q \arrow[dl, "f' \models c_Q"] \\
& G &
\end{tikzcd}$$

- $f \models \mathtt{true}$.
- $f \models \exists(a, c_Q)$ if there exists $f' : Q \to G$ such that $f' \circ a = f$ and $f' \models c_Q$.
- $f \models \neg c_P$ if $f \not\models c_P$.
- $f \models c_P \wedge c'_P$ if $f \models c_P$ and $f \models c'_P$.

As we may see, the models in our logic are not graphs but morphisms (roughly speaking, graphs extending the given context). In particular, we may consider that graphs are the models of conditions over the empty pattern (the initial pattern in the category).

The following lemma defines a construction called `Shift`, first introduced in [10, 11] under different conditions, to translate conditions along morphisms.

Lemma 1 (Shift of Conditions over Morphisms [9]) *Let* `Shift` *be a transformation of conditions inductively defined as follows:*

$$\begin{tikzcd}
P \arrow[r, "b"] \arrow[d, "a"'] \arrow[dr, phantom, "(1)"] & P' \arrow[d, "a'"] \\
Q \arrow[r, "b'"'] & Q' \\
\vartriangle & \vartriangle \\
c_Q & c_{Q'}
\end{tikzcd}$$

- $\mathtt{Shift}(b, \mathtt{true}) = \mathtt{true}$.
- $\mathtt{Shift}(b, \exists(a, c_Q)) = \exists(a', c_{Q'})$ *such that (1) is a pushout and* $c_{Q'} = \mathtt{Shift}(b', c_Q)$.
- $\mathtt{Shift}(b, \neg c_P) = \neg\mathtt{Shift}(b, c_P)$.
- $\mathtt{Shift}(b, c_P \wedge c'_P) = \mathtt{Shift}(b, c_P) \wedge \mathtt{Shift}(b, c'_P)$.

Then, for each condition c_P and each morphism $b : P \to P'$, $c_{P'} = \mathtt{Shift}(b, c_P)$ is a condition with context P' such that for each morphism $f : P' \to G$ we have that $f \models \mathtt{Shift}(b, c_P) \Leftrightarrow f \circ b \models c_P$.

4. An Institution for our Navigational Logic

The notion of *institution* was introduced in [5] as a conceptual tool to study constructions to structure and modularize specifications independently of any given formalism [13]. Let us recall its formal definition:

An *institution* [5] is a tuple of the form $\mathtt{I} = (\underline{\mathtt{Sig}}, \mathtt{Sen}, \mathtt{Mod}, \models)$ where,

- $\underline{\mathtt{Sig}}$ denotes the category of signatures of $\mathtt{I}$;
- $\mathtt{Sen} : \underline{\mathtt{Sig}} \to \underline{\mathtt{Set}}$ denotes the functor that maps every signature Σ into the set of all Σ-sentences, and every signature morphism $h : \Sigma_1 \to \Sigma_2$ into the mapping $\mathtt{Sen}(h)$ that translates Σ_1-sentences into Σ_2-sentences;
- $\mathtt{Mod} : \underline{\mathtt{Sig}} \to \underline{\mathtt{Cat}}^{\mathtt{op}}$ denotes the functor mapping every signature Σ into the category of all Σ-structures, and every signature morphism $h : \Sigma_1 \to \Sigma_2$ into its associated forgetful functor $\mathtt{V}_h = \mathtt{Mod}(h) : \mathtt{Mod}(\Sigma_2) \to \mathtt{Mod}(\Sigma_1)$;
- Finally, $\models$ is the *satisfaction relation of the institution* $\mathtt{I}$, consisting of a collection of relations $\models_\Sigma \subseteq Obj(Mod(\Sigma)) \times Obj(Sen(\Sigma))$ such that for every $h : \Sigma_1 \to \Sigma_2 \in \underline{\mathtt{Sig}}$, for every $A \in \mathtt{Mod}(\Sigma_2)$ and for every $\varphi \in \mathtt{Sen}(\Sigma_1)$ we have that
$$A \models_{\Sigma_2} \mathtt{Sen}(h)(\varphi) \Leftrightarrow \mathtt{Mod}(h)(A) \models_{\Sigma_1} \varphi$$

$\mathtt{I}$ is a *semi-exact* institution if $\underline{\mathtt{Sig}}$ has pushouts and, in addition, $\mathtt{Mod}$ transforms pushouts in $\underline{\mathtt{Sig}}$ into pullbacks in $\underline{\mathtt{Cat}}^{\mathtt{op}}$.

Let us now show that our navigational logic is an institution, which we call $\mathtt{NavLog}$. In our setting, we can identify signatures with patterns, in the sense that a pattern P determines the set of all sentences (conditions) of context P. Then, obviously, P-models are morphisms $P \to G$, where $G \in \underline{\mathtt{Graphs}}$, and satisfaction is defined as in Def. 2.

Proposition 1 (Institution $\mathtt{NavLog}$) *Given a category of patterns $\underline{\mathtt{Patterns}}$, together with its subcategory of graphs $\underline{\mathtt{Graphs}}$, we define the institution $\mathtt{NavLog} = (\underline{\mathtt{Sig}}, \mathtt{Sen}, \mathtt{Mod}, \models)$ as follows:*

1. **Signatures:** $\underline{\texttt{Sig}} = \underline{\texttt{Patterns}}$ *but, to avoid confusion, given a pattern P, we will denote the signature associated to P with* Σ_P .
2. **Sentences:** *Given a signature* Σ_P, $\texttt{Sen}(\Sigma_P)$ *is the set of all conditions with context P, as defined in Def. 2.*
   Given a signature morphism $h : \Sigma_{P_1} \to \Sigma_{P_2}$ *and a condition* $c_{P_1} \in \texttt{Sen}(\Sigma_{P_1})$, $\texttt{Sen}(h)(c_{P_1})$ *is defined as* $Sen(h)(c_{P_1}) = \texttt{Shift}(h, c_{P_1})$, *cf. Lemma 1 .*
3. **Models:** *Given a signature* Σ_P, *then* $Obj(\texttt{Mod}(\Sigma_P))$ *consists of all morphisms* $m : P \to G$, *where* $G \in \underline{\texttt{Graphs}}$.
   Given $m_1 : P \to G_1, m_2 : P \to G_2 \in Obj(\texttt{Mod}(\Sigma_P))$ *a morphism* $g : m_1 \to m_2 \in Morph(\texttt{Mod}(\Sigma_P))$ *is a morphism* $g : G_1 \to G_2 \in Morph(\underline{\texttt{Graphs}})$ *such that* $g \circ m_1 = m_2$.
$$\begin{tikzcd} P \arrow[rr, "m_1"] \arrow[dr, "m_2"'] & & G_1 \arrow[dl, "g"] \\ & G_2 & \end{tikzcd}$$
   Given a signature morphism $h : \Sigma_{P_1} \to \Sigma_{P_2}$ *and a* Σ_{P_2}*-model m,* $\texttt{V}_h(m)$, *is the morphism* $m \circ h$.
   And given a morphism $g : m_2 \to m_2'$ *in* $\texttt{Mod}(\Sigma_{P_2})$, *with* $m_2 : P_2 \to G_2$ *and* $m_2' : P_2 \to G_2'$ *(i.e.,* $g : G_2 \to G_2'$ *and* $g \circ m_2 = m_2'$*), we define* $\texttt{V}_h(g) = g$. *This definition is correct, since* $g \circ m_2 = m_2'$ *implies* $g \circ m_2 \circ h = m_2' \circ h$.
4. **Satisfaction relation:** *Satisfaction is defined as in Def. 2.*

Then, $\texttt{NavLog} = (\underline{\texttt{Sig}}, \texttt{Sen}, \texttt{Mod}, \models)$ *is an institution.*

PROOF. Consider any $h : \Sigma_{P_1} \to \Sigma_{P_2} \in \underline{\texttt{Sig}}$, any $m \in \texttt{Mod}(\Sigma_{P_2})$ and any $c_{P_1} \in \texttt{Sen}(\Sigma_{P_1})$, then as a consequence of Lem. 1, we have that $m \models_{\Sigma_{P_2}} \texttt{Shift}(h, c_{P_1}) \Leftrightarrow m \circ h \models_{\Sigma_{P_1}} c_{P_1}$ so, the satisfaction condition is directly satisfied. ■

From now on, we will write $\texttt{Mod}(c_P) = \{m \in \texttt{Mod}(\Sigma_P) \mid m \models_{\Sigma_P} c_P\}$. We can now see that NavLog is semi-exact:

Proposition 2 (NavLog **is semi-exact)** $\underline{\texttt{Sig}}$ *has pushouts if* $\underline{\texttt{Patterns}}$ *has pushouts.* Mod *transforms pushouts in* $\underline{\texttt{Sig}}$ *into pullbacks in* $\underline{\texttt{Cat}}^{\text{op}}$.

$$\begin{tikzcd} \Sigma_{P_0} \arrow[r, "h_2"] \arrow[d, "h_1"'] \arrow[dr, phantom, "p.o."] & \Sigma_{P_2} \arrow[d, "g_2"] & & \texttt{Mod}(\Sigma_{P_0}) \arrow[dr, phantom, "p.b."] & \texttt{Mod}(\Sigma_{P_2}) \arrow[l, "\texttt{V}_{h_2}"'] \\ \Sigma_{P_1} \arrow[r, "g_1"'] & \Sigma_P \arrow[r, phantom, ""] & {} \arrow[r, "\texttt{Mod}"] & {} & \\ & & & \texttt{Mod}(\Sigma_{P_1}) \arrow[uu, "\texttt{V}_{h_1}"] & \texttt{Mod}(\Sigma_P) \arrow[l, "\texttt{V}_{g_1}"] \arrow[uu, "\texttt{V}_{g_2}"'] \end{tikzcd}$$

PROOF. Since we assume that $\underline{\texttt{Patterns}}$ has pushouts, we have to prove that $\texttt{Mod}$ transforms pushouts in $\underline{\texttt{Sig}}$ into pullbacks in $\underline{\texttt{Cat}}^{\texttt{op}}$, i.e., that for every $\Sigma_{P'}$ such that $g'_1 : \Sigma_{P_1} \to \Sigma_{P'}$ and $g'_2 : \Sigma_{P_2} \to \Sigma_{P'}$ with $\texttt{V}_{h_1} \circ \texttt{V}_{g'_1} = \texttt{V}_{h_2} \circ \texttt{V}_{g'_2}$, there exists a unique $\texttt{V}_g : \texttt{Mod}(\Sigma_{P'}) \to \texttt{Mod}(\Sigma_P)$, such that $\texttt{V}_{g_1} \circ \texttt{V}_g = \texttt{V}_{g'_1}$ and $\texttt{V}_{g2} \circ \texttt{V}_g = \texttt{V}_{g'_2}$. First, since $\texttt{V}_{h_1} \circ \texttt{V}_{g'_1}(m') = \texttt{V}_{h_2} \circ \texttt{V}_{g'_2}(m')$ for every $m' : P' \to G \in Mod(\Sigma_{P'})$, we have that $m' \circ g'_1 \circ h_1 = m' \circ g'_2 \circ h_2$. Then, since the above square at the left is a pushout, we know there exists a unique $m : P \to G \in Mod(\Sigma_P)$ such that $m \circ g_1 = m' \circ g'_1$ and $m \circ g_2 = m' \circ g'_2$. That is, $\texttt{V}_{g_1}(m) = \texttt{V}_{g'_1}(m')$ and $\texttt{V}_{g2}(m) = \texttt{V}_{g'_2}(m')$. Therefore, $\texttt{V}_g(m') = m$ satisfies the required universal pullback property. ■

5. Structured Navigational Specifications and Structured Formulas

The proof that our navigational logic is a semi-exact institution allows us to build specifications using some standard specification building operations [13] or just using an algebraic specification language like CASL [2]. In particular, we can define generic models that can be instantiated to define more concrete ones. Or we may specify larger models by combining or extending simpler ones. For instance, if we want to model the information system of a company, on the one hand, we could specify its accounting system, perhaps by instantiating a generic accounting model. On the other hand, we could model the database of the employees of the company. Then we could combine the specification for the accounting system with the database model and extend the result with the description of the payroll.

However, in our context, structuring specifications may be not enough to make them readable, already single formulas may be too large to manage and to understand them. For instance, let us consider a condition like $\exists(a_0 : P_0 \to P_1, \forall(a_1 : P_1 \to P_2, \ldots \exists(a_k : P_k \to P_{k+1}, true) \ldots))$ [2] and let us suppose that $a_0, \ldots, a_k$ are just inclusions, which happens quite often, then patterns $P_0, \ldots, P_{k+1}$ would be of increasing size and P_{k+1} could be quite large.

For this reason, it is important to have operations that allow us to build large formulas from smaller ones. In our case, we have defined an operation called *Add* that given conditions $c_{P'}$ and $c_P = \sharp(a : P \to Q, c_Q)$ (where $\sharp$ denotes either $\exists$ or $\forall$)[3] and given a morphism $h : P' \to Q$, $Add(h, c_P, c_{P'})$ would add $c_{P'}$, translated through h, to c_P:

[2]In general, formulas may be more complex, because at each level of nesting we may have not just a *literal*, but an arbitrary formula involving several logical connectives.

[3]As usual, $\forall(a : P \to Q, c_Q)$ is an abbreviation for $\neg\exists(a : P \to Q, \neg c_Q)$

Proposition 3 (Addition) $\mathtt{Add} : \mathit{Morph}(\underline{\mathtt{Sig}}) \times \mathtt{Sen}(\Sigma_P) \times \mathtt{Sen}(\Sigma_{P'}) \to \mathtt{Sen}(\Sigma_P)$ *is defined for any morphism $h : P' \to Q$ and conditions $c_{P'}$ and $c_P = \sharp(a : P \to Q, c_Q)$, where $\sharp$ denotes a quantifier $\exists$ or $\forall$, as follows:*

$$\mathtt{Add}(h, c_P, c_{P'}) = \sharp(a : P \to Q, c_Q \wedge \mathtt{Sen}(h)(c_{P'}))$$

Then we have that, $\mathtt{Mod}(\mathtt{Add}(h, c_P, c_{P'})) = \mathtt{Mod}(c_P) \cap \mathtt{M}$ *where*

$$\mathtt{M} = \begin{cases} \{m \in \mathtt{Mod}(\Sigma_P) \mid \exists m' : m' \circ a = m \wedge \mathtt{V}_h(m') \in \mathtt{Mod}(c_{P'})\} & \textit{if } \sharp \textit{ is } \exists \\ \{m \in \mathtt{Mod}(\Sigma_P) \mid \forall m' : m' \circ a = m : \mathtt{V}_h(m') \in \mathtt{Mod}(c_{P'})\} & \textit{if } \sharp \textit{ is } \forall \end{cases}$$

PROOF. First of all recall that $Sen(h)(c_{P'}) = \mathtt{Shift}(h, c_{P'})$. If $m : P \to G$ such that $m \models \exists(a, c_Q \wedge \mathtt{Shift}(h, c_{P'}))$ then, by definition, this is equivalent to the existence of a morphism $m' : Q \to G$ such that $m = m' \circ a$ and $m' \models c_Q \wedge \mathtt{Shift}(h, c_{P'})$. Similarly, if $m : P \to G$ such that $m \models \forall(a, c_Q \wedge \mathtt{Shift}(h, c_{P'}))$ then, by definition, this is equivalent to $m' \models c_Q \wedge \mathtt{Shift}(h, c_{P'})$ for all $m' : Q \to G$ such that $m = m' \circ a$. Then, on the one hand, we have that $m \models c_P$ in both cases. On the other, recall that Lema 1 states that $m' \models_{\Sigma_{P'}} \mathtt{Shift}(h, c_{P'})$ if, and only if, $m' \circ h \models_{\Sigma_{P'}} c_{P'}$, and $\mathtt{V}_h(m') = m' \circ h$. Then, we can conclude the proof in both cases. ■

Let us see a simple example of the construction of a condition using *Add*. Suppose that we are specifying a social network represented by a graph, whose nodes may be labeled by names of persons or locations and having edges labeled by *friend*, *knows* or *visits*, and we want to state the following property:
"For any persons 1 *and* 2*, such that there is a path of friend edges from* 1 *to* 2*, if* 1 *and* 2 *may visit the same location then either they are the same person or* 1 *knows* 2*"*.

Let us consider the conditions depicted in Fig, 4. The first one, *JointLoc*, states that there should exist a location that may be visited by the given two persons. *Equal* states that the given two nodes are the same one, and *Knows* states that the person in node 1 should know the person in 2. Now, *Knows* and *Equal* are conditions with the same context P_{Two}, consisting of nodes 1 and 2, hence we may define the condition $Or = Knows \vee Equal$, also with context P_{Two}. Let us now define a morphism h from P_{Two} to the pattern on the right of condition *JointLoc*, mapping nodes 1 and 2 in P_{Two} into nodes 1 and 2 in the pattern in *JointLoc*. Then, $Add(h, JointLoc, Or)$ would specify that for two given nodes 1 and 2, if they may visit the same location, then either 1 knows 2 or they are the same person. Notice that the context of this condition is also P_{Two}. Finally, if h' is the morphism from

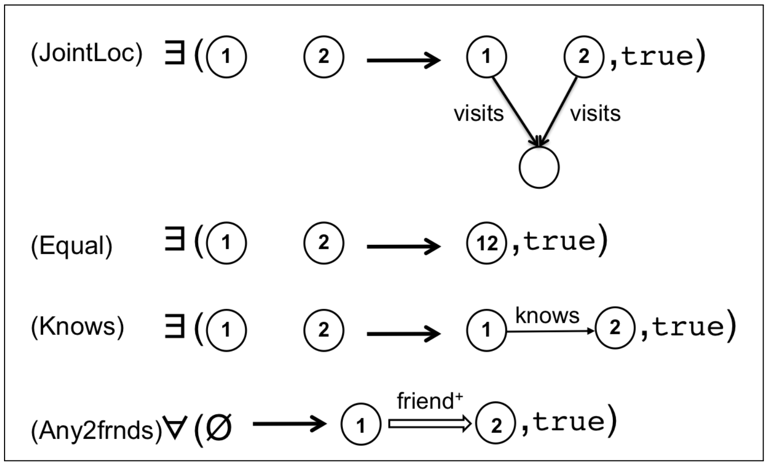


Figure 4: Some basic conditions

P_{Two} to the pattern on the right of $Any2frnds$, mapping nodes 1 and 2 in P_{Two} into nodes 1 and 2 in the pattern, then $Add(h', Any2frnds, Add(h, JointLoc, Or))$ would specify the property stated above.

6. Conclusion

In this paper we have shown that a navigational logic for the category of labeled graphs is a semi-exact institution, which allows us to structure and modularize our specifications as in [13]. Moreover, using the properties of our institution, we also show how to structure single formulas, which may be quite complex in our formalism. Even if, for simplicity, we have restricted our results to the case of labeled graphs, they can be directly generalized to arbitrary categories of graphical structures, following the lines of [8].

References

[1] Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J.L., Vrgoc, D.: Foundations of modern query languages for graph databases. ACM Comput. Surv. 50(5), 68:1–68:40 (2017)

[2] Astesiano, E., Bidoit, M., Kirchner, H., Krieg-Brückner, B., Mosses, P.D., Sannella, D., Tarlecki, A.: CASL: the common algebraic specification language. Theor. Comput. Sci. 286(2), 153–196 (2002)

[3] Barceló, P., Muñoz, P.: Graph logics with rational relations: The role of word combinatorics. ACM Trans. Comput. Log. 18(2), 10:1–10:41 (2017)

[4] Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Rozenberg, G. (ed.) Handbook of Graph Grammars. pp. 313–400. World Scientific (1997)

[5] Goguen, J., Burstall, R.: Institutions: Abstract model theory for specification and programming. Journal of the ACM 1(39), 95–149 (1992)

[6] Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Mathematical Structures in Computer Science 19(2), 245–296 (2009)

[7] Habel, A., Radke, H.: Expressiveness of graph conditions with variables. ECEASST 30 (2010)

[8] Navarro, M., Lambers, L., Orejas, F., Pino, E.: Towards a navigational logic for graphical structures. To be published. Festschrift in Memory of Hartmut Ehrig (2017)

[9] Navarro, M., Orejas, F., Pino, E., Lambers, L.: A logic of graph conditions extended with paths. In: Workshop on Graph Computation Models (GCM 2016), Vienna (2016)

[10] Pennemann, K.H.: Resolution-like theorem proving for high-level conditions. In: Graph Transformations, 4th International Conference, ICGT 2008. Lecture Notes in Computer Science, vol. 5214, pp. 289–304. Springer (2008)

[11] Pennemann, K.H.: Development of Correct Graph Transformation Systems, PhD Thesis. Department of Computing Science, Univ. of Oldenburg (2009)

[12] Poskitt, C.M., Plump, D.: Verifying monadic second-order properties of graph programs. In: Graph Transformation - 7th International Conference, ICGT 2014, Held as Part of STAF 2014, York, UK, July 22-24, 2014. Proceedings. pp. 33–48 (2014)

[13] Sannella, D., Tarlecki, A.: Specifications in an arbitrary institution. Inf. Comput. 76(2/3), 165–210 (1988)