Ensuring Correctness of Model Transformations While Remaining Decidable

Jon Haël Brenas, Rachid Echahed, Martin Strecker

Ensuring Correctness of Model Transformations while Remaining Decidable *

Jon Haël Brenas[1], Rachid Echahed[1], and Martin Strecker[2]

[1] CNRS and Université de Grenoble Alpes
[2] Université de Toulouse / IRIT

Abstract. This paper is concerned with the interplay of the expressiveness of model and graph transformation languages, of assertion formalisms making correctness statements about transformations, and the decidability of the resulting verification problems. We put a particular focus on transformations arising in graph-based knowledge bases and model-driven engineering. We then identify requirements that should be satisfied by logics dedicated to reasoning about model transformations, and investigate two promising instances which are decidable fragments of first-order logic.

Keywords: Graph Transformation, Model Transformation, Program Verification, Classical Logic, Modal Logic

1 Introduction

We tackle the problem of model transformations and their correctness, where transformations are specified with the aid of rules and correctness properties are stated as logical formulas. By model we intend a graph structure enriched with logical formulas which label either nodes or edges. In our approach, a rule is composed of a left-hand side which is a graph annotated with logical formulas, and a right-hand side which is a sequence of actions. The shape of the graph and the formulas yield an applicability condition of the rule at a matching subgraph of the model; the right-hand side transforms this subgraph with actions such as creation, deletion or cloning of nodes or insertion and deletion of arcs.

Rewrite systems come with a specification in the form of pre- and postconditions, and we aim at full deductive verification, ascertaining that any model satisfying the precondition is transformed into a model satisfying the postcondition.

The correctness of model transformations has attracted some attention in the last years. One prominent approach is model checking, such as implemented by the Groove tool [13]. The idea is to carry out a symbolic exploration of the state space, starting from a given model, in order to find out whether certain invariants are maintained or certain states (*i.e.* , model configurations) are reachable. The

* This research has been supported by the *Climt* project (ANR-11-BS02-016).

Viatra tool has similar model checking capabilities [25] and in addition allows the verification of elaborate well-formedness constraints imposed on models [23]. Well-formedness is within the realm of our approach (and amounts to checking the consistency of a formula), but is not the primary goal of this paper which is on the dynamics of models.

The Alloy analyser [17] uses bounded model checking for exploring relational designs and transformations (see for example [5] for an application in graph transformations). Counter-examples are presented in graphical form. All the aforementioned techniques use powerful SAT- or SMT-solvers, but do not carry out a complete deductive verification. In our paper, we aim at full-fledged verification of transformations.

General-purpose program verification with systems such as AutoProof [24] and Dafny [18] becomes increasingly automated and thus interesting as push-button technology for model transformations. In this context, fragments of first-order logic have been proposed that are decidable and are useful for dealing with pointer structures [16].

The question explored in this paper is: which requirements does a logic have to fulfill in order to allow for such a verification technique to succeed?

Several different logics have been proposed over the years to tackle the problem of graph transformation verification. Among the most prominent approaches figure nested conditions [15,20] that are explicitly created to describe graph properties. Another widely used logic in graph transformation verification is monadic second-order logic [10,21] that allows to go beyond first-order definable properties. [4] introduces a logic closer to modal logic that allows to express both graph properties and the transformations at the same time.

Nonetheless, these approaches are not flawless. They are all undecidable in general and thus either cannot be used to prove correctness of graph transformations in an automated way or only work on limited classes of graphs. Starting from the other side of the logical spectrum, one could consider using Description Logics to describe graph properties [1,6] that are decidable. Another choice could be the use of modal logics as they are suited to reason about programs. Obviously, this comes at a cost in term of expressiveness.

Separation logic [22] is another choice that is worth considering when dealing with transformations of graphs. It has been developed especially to be able to talk about pointers in conventional programming languages.

In this paper, we proceed in an orthogonal direction. Instead of introducing a logic and advising users to tailor their problem so that it is expressible in our logic and that its models comply with the restrictions so that the verification is actually possible, we aim at providing a means for the users to decide whether the logic they have used to represent their problem will actually allow them to prove their transformations correct or whether they have to use several different systems in parallel.

We are in particular interested in decidable logics, and so we instantiate our general framework with two decidable logics: Two-variable logic with counting (in Section 5.1) and logics with exists-forall-prefix (in Section 5.2). The fragment

of effectively propositional logic [19], that is implemented by the Z3 prover [11] and is closely related to the logical fragment we discuss in Section 5.2, has been known for a long time to be decidable [8]. The use of two-variable logics [14] for the verification of model transformation is relatively novel even though it contains all Description Logics without role inclusions. Once more the goal is not to advocate the use of any logic but to give the user the ability to decide if the logics that are planned to be used satisfy some minimal conditions so that the verification can be carried out effectively.

The rest of the paper is structured as follows: we start with an example, in Section 2, motivating our model transformation approach, which we then make more formal in Section 3. In Section 4, we propose general principles that a logic has to fulfill to be usable for verifying model transformations. Then, in Section 5, we illustrate our proposal through the two aforementioned logics. Concluding remarks are provided in Section 6.

2 Motivating Example

In order to better illustrate our purpose, an example modelling a sample of the information system of a hospital is introduced. Figure 1 is the UML model of this sample.

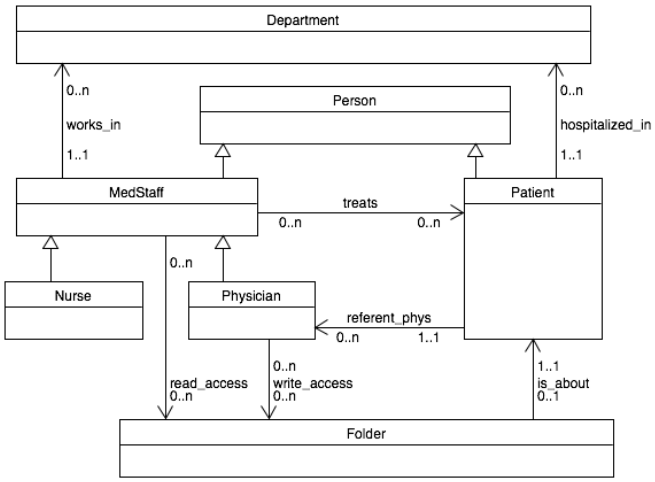

Fig. 1: A sample UML model for the hospital example

We consider persons (shortened to PE). Some of them work in the hospital and form the medical staff (MS) and others are patients (PA). The medical staff is partitioned into physicians (PH) and nurses (NU). In addition, the hospital

is split into several departments (DE) or services. Documents pertaining to patients are stored in folders (FO).

Each member of the medical staff is assigned (denoted by *works_in*) to a department. The same way, each patient is hospitalized (*hospitalized_in*) in one of the departments. There may be several members of the medical staff that may collaborate to treat (*treats*) a patient at a given time but one of them is considered as the referent physician (*referent_phys*), that is to say she is in charge of the patient. Part of the medical staff can access the folder containing the documents about (*is_about*) a patient either to read (*read_access*) or to write (*write_access*) information.

The fact is the hospital is bound to evolve: new patients arrive to be cured and others leave, new medical staffers are hired and others move out. To illustrate our purpose, four possible transformations are specified below.

Transformation 1 *The first transformation is New_Ph(*PH_1, D_1*). It creates a new physician to which is associated an identifier* PH_1*. This physician will be working in the department identified with* D_1*.*

Transformation 2 *The second transformation is New_Pa(*PA_1, PH_1, FO_1*). It adds a new patient. The patient* PA_1 *is created alongside his folder* FO_1*. She is then assigned* PH_1 *as referent physician.*

Transformation 3 *The third transformation is Del_Pa(*PA_1*). It modifies data so that patient* PA_1*is no more hospitalized.*

Transformation 4 *The last transformation is Del_Ph(*PH_1, PH_2*). It deletes the physician* PH_1 *and forwards all his patients to the physician* PH_2*.* PH_1 *and* PH_2 *have to work in the same department.*

Despite the transformations, there are some properties of the hospital that should not be altered. We give a list of six such expected properties in the following.

Expected property 1 *Each member of the medical staff is either a nurse or a physician but not both.*

Expected property 2 *All patients and all medical staffers are persons.*

Expected property 3 *Each person that can write in a folder can also read it.*

Expected property 4 *Each person that can read a folder about a patient treats that patient.*

Expected property 5 *Only medical staffers can treat persons and only patients can be treated.*

Expected property 6 *Every patient has exactly one referent physician.*

3 A Model Transformation Framework

In this section, a framework used to describe models as well as their transformations is introduced. A model is considered hereafter as a graph, labeled by logical formulae. The logic in which these formulae are expressed is considered as a parameter, say $\mathcal{L}$, of the proposed framework. Required features of such a logic are discussed in the next section. Nevertheless, we assume in this section that the logic $\mathcal{L}$ is endowed with a relation $\models$ over its formulae. That is to say, $n \models B$ (resp. $e \models B$) should be understood as "formula B is satisfied at node n (resp. edge e)".

Definition 1 (Graph). *Let $\mathcal{L}$ be a logic. A* graph *G is a tuple (N, E, $\mathcal{C}$, $\mathcal{R}$, ϕ_N, ϕ_E, s, t) where N is a set of* nodes*, E is a set of* edges*, $\mathcal{C}$ is a set of* (node) formulae *(of $\mathcal{L}$) or* concepts*, $\mathcal{R}$ is the set of* edge formulae *(of $\mathcal{L}$) or* roles*, ϕ_N is the* node labeling *function, $\phi_N : N \to \mathcal{P}(\mathcal{C})$, ϕ_E is the* edge labeling *function, $\phi_E : E \to \mathcal{R}$, s is the* source function *$s : E \to N$ and t is the* target function *$t : E \to N$.*

Labeling a graph with logical formulae is quite usual in Kripke structures. In this paper, labeling formulae will play a role either in the transformation process or in the generation of proof obligations for the properties intended to be proved.

Transformations of models are performed by means of graph rewrite systems. These rewrite systems are extensions of those defined in [12] where graphs are labeled with formulae. Thus, the left-hand sides of the rules are labeled graphs as defined in Definition 1, whereas the right-hand sides are defined as sequences of elementary actions. Elementary actions constitute a set of basic transformations used in graph transformation processes. They are given in the following definition.

Definition 2 (Elementary action, action). *An* elementary action*, say a, has one of the following forms:*

- *a* concept assignment *$c := i$ where i is a node and c is an atomic formula (a unary predicate). It sets the valuation of c such that the only node labeled by c is i.*
- *a* concept addition *$c := c + i$ (resp.* concept deletion *$c := c - i$) where i is a node and c is an atomic formula (a unary predicate). It adds the node i to (resp. removes the node i from) the valuation of the formula c.*
- *a* role addition *$r := r + (i, j)$ (resp.* role deletion *$r := r - (i, j)$) where i and j are nodes and r is an atomic role (a binary predicate). It adds the pair (i, j) to (resp. removes the pair (i, j) from) the valuation of the role r.*
- *a* node addition *$new(i)$ (resp.* node deletion *$del_I(i)$) where i is a new node (resp. an existing node). It creates the node i. i has no incoming nor outgoing edge and there is no atomic formula such that i belongs to its valuation (resp. it deletes i and all its incoming and outgoing edges).*
- *a* global incoming edge redirection *$i \gg^{in} j$ where i and j are nodes. It redirects all incoming edges of i towards j.*

– *a* global outgoing edge redirection $i \gg^{out} j$ *where* i *and* j *are nodes. It redefines the source of all outgoing edges of* i *as* j.
– *a* node cloning $clone(i, i')$ *where* i *is a node,* i' *is a node that does not exist yet. It creates a new node* i' *that has the same labels as* i *and the same incoming and outgoing edges*[3].

The result of performing the elementary action α *on a graph* $G = (N^G, E^G, \mathcal{C}^G, \mathcal{R}^G, \phi_N^G, \phi_E^G, s^G, t^G)$ *produces the graph* $G' = (N^{G'}, E^{G'}, \mathcal{C}^{G'}, \mathcal{R}^{G'}, \phi_N^{G'}, \phi_E^{G'} s^{G'}, t^{G'})$ *as defined in Figure 2 and write* $G' = G[\alpha]$ *or* $G \Rightarrow_\alpha G'$. *An* action, *say* α, *is a sequence of elementary actions of the form* $\alpha = a_1; a_2; \ldots; a_n$. *The result of performing* α *on a graph* G *is written* $G[\alpha]$. $G[a; \alpha] = (G[a])[\alpha]$ *and* $G[\epsilon] = G$, ϵ *being the empty sequence.*

Definition 3 (Rule, Graph Rewrite Systems). *A* rule $\rho[\boldsymbol{n}]$ *is a pair* (lhs, α) *where* $\boldsymbol{n}$ *is a vector of concept variables. These variables are instantiated by means of actual concepts when a rule is applied.* lhs, *called the left-hand side, is a graph and* α, *called the right-hand side, is an action. Rules are usually written* $\rho[\boldsymbol{n}] : lhs \to \alpha$. *Concept variables* n_i *in* $\boldsymbol{n}$ *may appear both in* lhs *and in* α. *A graph rewrite system is a set of rules.*

Notice that a rule $\rho[\boldsymbol{n}] : lhs \to \alpha$ may be considered as a generic rule which yields an actual rewrite rule for every instance of the variables $\boldsymbol{n}$. We write $\rho[\boldsymbol{c}]$ to denote the rule obtained from $\rho[\boldsymbol{n}] : lhs \to \alpha$ by replacing every variable concept n_i appearing either in lhs or in α by the actual concept c_i. Now let us define when a rule can be applied to a graph.

Definition 4 (Match). *Let* $\rho[\boldsymbol{n}] : lhs \to \alpha$ *be a rule and* G *be a graph. Let* $\rho[\boldsymbol{c}]$ *be an instance of rule* $\rho[\boldsymbol{n}]$ *and* $inst$ *be the instance function defined as* $inst(n_i) = c_i$ *for* $i \in \{0, \ldots, k\}$. *We say that the instance* $\rho[\boldsymbol{c}]$ *matches the graph* G *via the* match $h = (h_N, h_E)$, *where* $h_N : N^{lhs} \to N^G$ *and* $h_E : E^{lhs} \to E^G$ *if the following conditions hold:*

1. $\forall n \in N^{lhs}, \forall d \in \phi_{N_{lhs}}(n), h_N(n) \models inst(d)$
2. $\forall e \in E^{lhs}, \forall r \in \phi_{E_{lhs}}(e), h_E(e) \models inst(r)$[4]
3. $\forall e \in E^{lhs}, s_G(h_E(e)) = h_N(s_{lhs}(e))$
4. $\forall e \in E^{lhs}, t_G(h_E(e)) = h_N(t_{lhs}(e))$

The third and the fourth conditions are classical and say that the source and target functions and the match have to agree. The first condition says that for every node n of the left-hand side, the node to which it is associated, $h_N(n)$, in G has to satisfy every concept that n satisfies. This condition clearly expresses additional negative and positive conditions which are added to the "structural" pattern matching. The second condition expresses the same conditions on the edges.

[3] This action has the same effect as the one defined by means of sesquipushout [9].

[4] $inst(r)$ (resp. $inst(d)$) replaces in r (resp. in d) every occurrence of a concept variable n_i by its instance c_i. The formal definition of the function $inst$ depends on the structure of the considered concepts and roles.

If $\alpha = c := i$ **then:**
$N^{G'} = N^G, E^{G'} = E^G, \mathcal{C}^{G'} = \mathcal{C}^G, \mathcal{R}^{G'} = \mathcal{R}^G$
$\phi_N^{G'}(n) = \begin{cases} \phi_N^G(n) \cup \{c\} \text{ if } n = i \\ \phi_N^G(n) \backslash \{c\} \quad \text{if } n \neq i \end{cases}, \phi_E^{G'} = \phi_E^G,$
$s^{G'} = s^G,\ t^{G'} = t^G$

If $\alpha = c := c + i$ **then:**
$N^{G'} = N^G, E^{G'} = E^G, \mathcal{C}^{G'} = \mathcal{C}^G, \mathcal{R}^{G'} = \mathcal{R}^G,$
$\phi_E^{G'} = \phi_E^G, \phi_N^{G'}(n) = \begin{cases} \phi_N^G(n) \cup \{c\} \text{ if } n = i \\ \phi_N^G(n) \qquad \text{if } n \neq i \end{cases}$
$s^{G'} = s^G,\ t^{G'} = t^G$

If $\alpha = c := c - i$ **then:**
$N^{G'} = N^G, E^{G'} = E^G, \mathcal{C}^{G'} = \mathcal{C}^G, \mathcal{R}^{G'} = \mathcal{R}^G,$
$\phi_E^{G'} = \phi_E^G,\ \phi_N^{G'}(n) = \begin{cases} \phi_N^G(n) \backslash \{c\} \text{ if } n = i \\ \phi_N^G(n) \qquad \text{if } n \neq i \end{cases}$
$s^{G'} = s^G,\ t^{G'} = t^G$

If $\alpha = r := r + (i, j)$ **then** :
$N^{G'} = N^G, \mathcal{C}^{G'} = \mathcal{C}^G, \mathcal{R}^{G'} = \mathcal{R}^G,$
$E^{G'} = E^G \cup \{e\}$ where e is a new element
$\phi_N^{G'} = \phi_N^G,\ \phi_E^{G'}(e') = \begin{cases} r & \text{if } e' = e \\ \phi_E^G(e') & \text{if } e' \neq e \end{cases},$
$s^{G'}(e') = \begin{cases} i & \text{if } e' = e \\ s^G(e') & \text{if } e' \neq e \end{cases},$
$t^{G'}(e') = \begin{cases} j & \text{if } e' = e \\ t^G(e') & \text{if } e' \neq e \end{cases}$

If $\alpha = r := r - (i, j)$ **then:**
$N^{G'} = N^G, \mathcal{C}^{G'} = \mathcal{C}^G, \mathcal{R}^{G'} = \mathcal{R}^G$
$E^{G'} = E^G \backslash r_{i,j},$ where
$r_{i,j} = \{e \in E^G | s^G(e) = i \wedge t^G(e) = j \wedge \phi_E^G(e) = r\}$
$\phi_N^{G'} = \phi_N^G, \phi_E^{G'}$ is the restriction of ϕ_E^G to $E^{G'}$
$s^{G'}$ is the restriction of s^G to $E^{G'}$
$t^{G'}$ is the restriction of t^G to $E^{G'}$

If $\alpha = new(i)$ **then:**
$N^{G'} = N^G \cup \{i\}$ where i is a new node,
$E^{G'} = E^G, \mathcal{C}^{G'} = \mathcal{C}^G, \mathcal{R}^{G'} = \mathcal{R}^G,$
$\phi_N^{G'}(n) = \begin{cases} \emptyset & \text{if } n = i \\ \phi_N^G(n') & \text{if } n \neq i \end{cases}$
$\phi_E^{G'} = \phi_E^G,\ s^{G'} = s^G,\ t^{G'} = t^G$

If $\alpha = del(i)$ **then:**
$N^{G'} = N^G \backslash \{i\}, \mathcal{C}^{G'} = \mathcal{C}^G, \mathcal{R}^{G'} = \mathcal{R}^G,$
$E^{G'} = E^G \backslash \{e | s^G(e) = i \vee t^G(e) = i\}$
$\phi_N^{G'}$ is the restriction of ϕ_N^G to $N^{G'}$
$\phi_E^{G'}$ is the restriction of ϕ_E^G to $E^{G'}$
$s^{G'}$ is the restriction of s^G to $E^{G'}$
$t^{G'}$ is the restriction of t^G to $E^{G'}$

If $\alpha = i \gg^{in} j$ **then** :
$N^{G'} = N^G, E^{G'} = E^G, \mathcal{C}^{G'} = \mathcal{C}^G,$
$\mathcal{R}^{G'} = \mathcal{R}^G, \phi_N^{G'} = \phi_N^G, \phi_E^{G'} = \phi_E^G,$
$s^{G'} = s^G, t^{G'}(e) = \begin{cases} j & \text{if } t^G(e) = i \\ t^G(e) & \text{if } t^G(e) \neq i \end{cases}$

If $\alpha = i \gg^{out} j$ **then:**
$N^{G'} = N^G, E^{G'} = E^G, \mathcal{C}^{G'} = \mathcal{C}^G,$
$\mathcal{R}^{G'} = \mathcal{R}^G, \phi_N^{G'} = \phi_N^G, \phi_E^{G'} = \phi_E^G,$
$\phi_N^{G'} = \phi_N^G, t^{G'} = t^G,$
$s^{G'}(e) = \begin{cases} j & \text{if } s^G(e) = i \\ s^G(e) & \text{if } s^G(e) \neq i \end{cases}$

If $\alpha = clone(i, i')$ **then:**
$\mathcal{C}^{G'} = \mathcal{C}^G, \mathcal{R}^{G'} = \mathcal{R}^G$
$N^{G'} = N^G \cup \{i'\},\ E^{G'} = E^G \cup E_i'$ where
$E_i' = E_i^{in} \cup E_i^{out} \cup E_i^{loop}$ with
$E_i^{in} = \{e^{in} | \ \exists e \in E^G, t^G(e) = i\}$
$E_i^{out} = \{e^{out} | \ \exists e \in E^G, s^G(e) = i\}$
$E_i^{loop} = \{e^{loop} | \exists e \in E^G, s^G(e) = t^G(e) = i\}$
$\phi_N^{G'}(n) = \begin{cases} \phi_N^G(n) \text{ if } n \neq i' \\ \phi_N^G(i) \text{ otherwise} \end{cases}$
$\phi_E^{G'}(e) = \begin{cases} \phi_E^G(e) \text{ if } e \notin E_i' \\ \phi_E^G(co(e)) \text{ otherwise} \end{cases}$
$t^{G'}(e) = \begin{cases} t^G(e) \text{ if } e \notin E_i' \\ t^G(co(e)) \text{ if } e \in E_i^{out} \\ i' \text{ if } e \in E_i^{in} \cup E_i^{loop} \end{cases}$
$s^{G'}(e) = \begin{cases} s^G(e) \text{ if } e \notin E_i' \\ s^G(co(e)) \text{ if } e \in E_i^{in} \\ i' \text{ if } e \in E_i^{out} \cup E_i^{loop} \end{cases}$
where for $e' \in E'$, $co(e')$ is the edge e that e' is a copy of.

Fig. 2: Summary of the effects of atomic actions

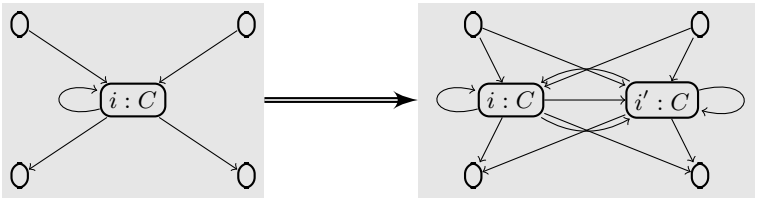

Fig. 3: Example of node cloning. The action $clone(i, i')$ is performed.

Definition 5 (Rule application). *We define the applicability condition as:* $App(\rho[\boldsymbol{c}], G)$ *iff there exists a match* h *from the instance* $\rho[\boldsymbol{c}]$ *to* G*. A graph* G *rewrites to graph* G' *using a rule* $\rho[\boldsymbol{c}] : lhs \to \alpha$ *iff* $App(\rho[\boldsymbol{c}], G)$ *holds and* G' *is obtained from* G *by performing actions in* $h(\alpha)$[5]*. Formally,* $G' = G[h(\alpha)]$*. We write* $G \to_{\rho[\boldsymbol{c}]} G'$ *or* $G \to_{\rho[\boldsymbol{c}],h} G'$*.*

Example 1. Let us consider again the example given in Section 2. We provide in Figure 4, for every transformation already presented informally, a corresponding rewrite rule.

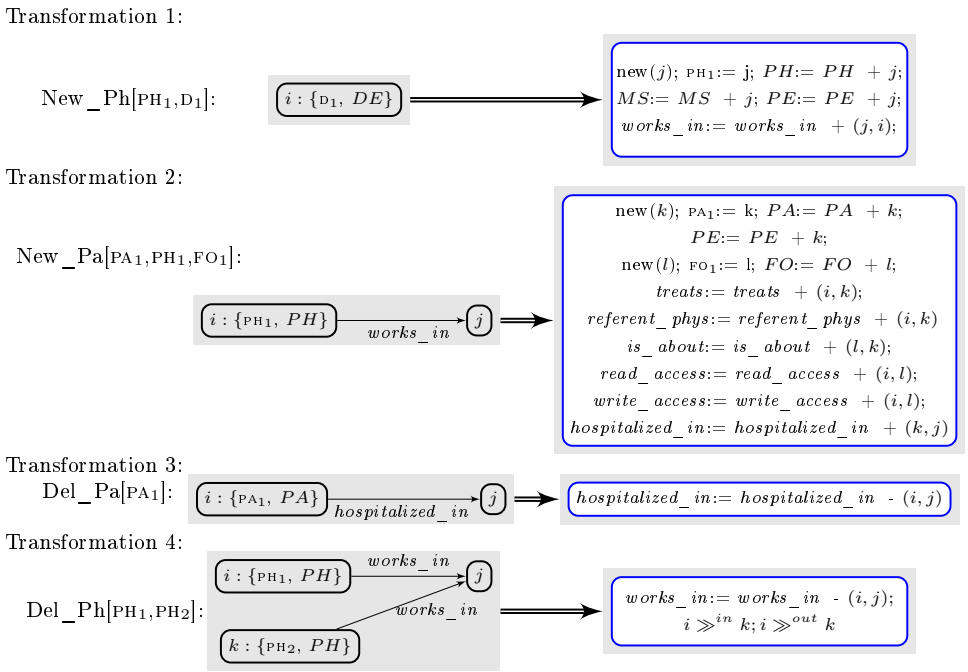

Fig. 4: Transformation rules for the sample hospital model

[5] $h(\alpha)$ is obtained from α by replacing every node name, n, of lhs by $h(n)$.

Very often, transforming models by means of rewrite rules necessitates the use of the notion of strategies. Informally, a strategy acts as a recipe indicating in which order the rules are applied.

Definition 6 (Strategy). *Given a graph rewriting system $\mathcal{R}$, a* strategy *is a word of the following language defined by s:*

$s := \rho[c_0, \ldots, c_k]$ *(Rule application)* $\mid s^*$ *(Closure)*

$\quad s; s$ *(Composition)* $\mid s \oplus s$ *(Choice)*

where $\rho[c_0, \ldots, c_k]$ is an instance of a rule in $\mathcal{R}$.

We write $G \Rightarrow_{\mathcal{S}} G'$ when G rewrites to G' following the rules given by the strategy $\mathcal{S}$.

Informally, the strategy "$\rho_1; \rho_2$" means that rule ρ_1 should be applied first, followed by the application of rule ρ_2. Notice that the strategies as defined above allow one to define infinite derivations from a given graph G because we have included the Kleene star construct s^* as a constructor of strategies. Handling the Kleene star does not introduce much more difficulties but requires the use of the notion of invariants in the verification procedures, as it is the case for while loops in imperative languages. It also requires us to extend the notion of applicability from rules to strategies:

$\mathrm{App}(s^*, G) = \mathrm{true}$ $\qquad$ $\mathrm{App}(s_0; s_1, G) = \mathrm{App}(s_0, G)$

$\mathrm{App}(s_0 \oplus s_1, G) = \mathrm{App}(s_0, G) \vee \mathrm{App}(s_1, G)$

In Figure 5, we provide the rules that specify how strategies are used to rewrite a model (graph). Notice that a closure free strategy is always terminating while a choice free strategy is always confluent.

(Rule application) $\dfrac{G \rightarrow_{\rho[\mathbf{c}]} G'}{G \Rightarrow_{\rho[\mathbf{c}]} G'}$ $\qquad$ (Choice left) $\dfrac{G \Rightarrow_{s_0} G'}{G \Rightarrow_{s_0 \oplus s_1} G'}$ $\qquad$ (Choice right) $\dfrac{G \Rightarrow_{s_1} G'}{G \Rightarrow_{s_0 \oplus s_1} G'}$

(Composition) $\dfrac{G \Rightarrow_{s_0} G'' \quad G'' \Rightarrow_{s_1} G'}{G \Rightarrow_{s_0; s_1} G'}$ $\qquad$ (Closure applicable) $\dfrac{G \Rightarrow_{s} G'' \; G'' \Rightarrow_{s^*} G' \; App(s, G)}{G \Rightarrow_{s^*} G'}$ $\qquad$ (Closure Inapplicable) $\dfrac{\neg App(s, G)}{G \Rightarrow_{s^*} G}$

Fig. 5: Strategy application rules

To end this section we define the notion of a specification which consists in providing *Pre* and *Post* conditions that one may want to ensure for a given strategy. More precisely, we propose the following definitions.

Definition 7 (Program, Specification). *A* program *is a tuple $(\mathcal{R}, \mathcal{S})$ where $\mathcal{R}$ is a graph rewrite system and $\mathcal{S}$ is a strategy. A* specification *SP is a tuple $(Pre, Post, \mathcal{P})$ where Pre and $Post$ are formulae and $\mathcal{P}$ is a program.*

Notice that Pre and $Post$ are supposed to be formulae of a given logic. We do not specify such a logic in the above definition. We provide actual examples in Section 5. A specification $(Pre,\ Post,\ \mathcal{P})$ asserts that for all models G that satisfies the formula Pre, all models G' obtained after rewriting G according to strategy $\mathcal{S}$ of program $\mathcal{P} = (\mathcal{R}, \mathcal{S})$, (i.e. $G \Rightarrow_{\mathcal{S}} G'$), G' satisfies formula $Post$.

4 General Logical Framework

Our aim in this section is to discuss general requirements for a logic, say $\mathcal{L}$, that might be considered either to specify pre and post conditions of specifications or to label models.

Let $SP = (Pre,\ Post,\ \mathcal{P})$ be a specification. If SP is correct, then if a model G satisfies Pre ($G \models Pre$) and G rewrites to model G' via a strategy $\mathcal{S}$ of a program $\mathcal{P} = (\mathcal{R}, \mathcal{S})$ ($G \Rightarrow_{\mathcal{S}} G'$), then G' satisfies $Post$ ($G' \models Post$). In addition to the general requirements for logics $\mathcal{L}$, a Hoare-like calculus dedicated to prove the correctness of specifications is also discussed in this section .

The first, and most obvious, requirements for a logic, $\mathcal{L}$, is that it can express the labeling of models with formulae which specify nodes and edges.

Requirement 1 *Node formulae (concepts in $\mathcal{C}$) should be adequate to the notion of nodes. That is to say, nodes might be candidates to interpret node formulae.*

Requirement 2 *Edge formulae (roles in $\mathcal{R}$) should be adequate to the notion of edges. That is to say, edges might be candidates to interpret edge formulae.*

The conditions Pre and $Post$ are properties of models. Thus, we have the following requirement.

Requirement 3 *Assertions Pre and Post should be adequate to the notion of graphs (i.e. models). That is to say, models might be candidates to interpret Pre and Post assertions.*

The main ingredient of the verification calculus consists in computing weakest preconditions of postconditions (see function wp defined in Fig 6). The basic cases of the computations of weakest precondition deal with elementary actions. For that, to every elementary action is associated a so called substitution. Such substitutions are the elementary building blocks allowing the verification of a program.

Definition 8. *Let a be an elementary action, as defined in Definition 2. The substitution $[a]$ associated to the elementary action a is the formula constructor which associates, to each formula ϕ of $\mathcal{L}$, the formula $\phi[a]$. Given a model $\mathcal{M}$, $\phi[a]$ is defined such that $\mathcal{M} \models \phi[a] \Leftrightarrow$ for all models $\mathcal{M}', \mathcal{M} \Rightarrow_a \mathcal{M}'$ implies $\mathcal{M}' \models \phi$.*

A logic $\mathcal{L}'$ is said to be closed under substitutions if for each action a, for each formula ϕ of $\mathcal{L}'$, $\phi[a]$ is also a formula of $\mathcal{L}'$.

$$wp(\rho[\boldsymbol{c}],\ Q) = App(tag(\rho[\boldsymbol{c}])) \Rightarrow wp(tag(\alpha_{\rho[\boldsymbol{c}]}), Q)$$
$$wp(s_0; s_1,\ Q) = wp(s_0,\ wp(s_1,\ Q)) \qquad wp(s^*) = inv_s$$
$$wp(s_0 \oplus s_1,\ Q) = wp(s_0, Q) \wedge wp(s_1, Q)$$

Fig. 6: Weakest preconditions for strategies.

$$vc(\rho[\boldsymbol{c}],\ Q) = \text{true} \qquad vc(s_0; s_1,\ Q) = vc(s_0,\ wp(s_1,\ Q)) \wedge vc(s_1, Q)$$
$$vc(s_0 \oplus s_1,\ Q) = vc(s_0, Q) \wedge vc(s_1, Q)$$
$$vc(s^*,\ Q) = (inv_s \wedge App(s) \Rightarrow wp(s, inv_s)) \wedge (inv_s \wedge \neg App(s) \Rightarrow Q) \wedge vc(s, inv_s) \wedge vc(s_1, Q)$$

Fig. 7: Verification conditions for strategies.

Weakest preconditions for actions come in two flavors: for elementary actions a, we have $wp(a, Q) = Q[a]$, and for composite actions, $wp(a; \alpha,\ Q) = wp(a,\ wp(\alpha,\ Q))$. On this basis, weakest preconditions for strategies can be easily computed as depicted in Figure 6. These preconditions follow the principles of Hoare Logic calculi except for the one dedicated to rules, viz. $wp(\rho[\boldsymbol{c}],\ Q)$. This latter corresponds essentially to an "if-then" structure in imperative programs. Put it simply, it checks three properties that are required for the application of a rule to be correct. Up to now, *App* depended on G. However, correctness proofs should hold for all possible models (graphs). That is way we modify *App* to be dependent only on the rules and strategies. First, *App* is a function which applies to a rule $\rho[\boldsymbol{c}]$ and returns a formula of $\mathcal{L}$ stating that there exists a match from the left-hand side of $\rho[\boldsymbol{c}]$ to a potential graph. If the formula $App(\rho[\boldsymbol{c}])$ is satisfied, the rule can be performed. Second, whenever the formula $App(\rho[\boldsymbol{c}]) \Rightarrow wp(\alpha_{\rho[\boldsymbol{c}]}, Q)$ is valid, then if there exists a match, the conditions, viz. $wp(\alpha_{\rho[\boldsymbol{c}]}, Q)$, which ensure the postcondition to be satisfied, are satisfied too. This corresponds to the usual weakest-precondition in Hoare Logic.

There is one additional issue which deserves to be handled carefully. Actually, one same rule can be fired several times during the execution of a program. It is thus mandatory to keep track of where each occurence of the rule is applied. To be more precise, *App* introduces a condition that uses the names of the nodes in the left-hand sides of rules. As these names uniquely define nodes and edges, if a same rule were used several times with the same names of nodes and edges, the rule would be applied to the exact same nodes and edges. This issue is solved by renaming the individuals (i.e., nodes and edges) each time the rule is fired. This is done through the function *tag*. That is why $wp(\rho[\boldsymbol{c}], Q) = App(tag(\rho[\boldsymbol{c}])) \Rightarrow wp(tag(\alpha_{\rho[\boldsymbol{c}]}), Q)$.

Finally, the closure of a strategy, s^*, which is close to while structures in imperative programs, needs the definition of an invariant, inv_s, and the intro-

duction of verification conditions, $vc(s^*, Q)$, shown in Figure 7. Basically, the idea is that a closure is considered as a subprogram whose correctness is proven on the side. The verification condition checks that the specification of this subprogram whose pre and post conditions are the invariant.

From the discussion above, we come to a new requirement about the logic $\mathcal{L}$, regarding the use of substitutions within weakest preconditions.

Requirement 4 *$\mathcal{L}$ must be closed under substitutions.*

If this last requirement is not satisfied, the computation of weakest preconditions may lead to formulas not expressible in $\mathcal{L}$. In this case, the verification of the correctness of specifications would need new proof procedures different from those of $\mathcal{L}$.

In addition, $App(\rho[\boldsymbol{c}])$ must be definable in $\mathcal{L}$. Obviously, this depends mainly on the rules one wants to use. It is thus possible, for a given problem, to use one logic that may not be powerful enough for other problems. Nonetheless, one of the requirements this entails on $\mathcal{L}$ is that it must allow some kind of existential quantification so that the graph can be traversed to look for a match. Obviously, the $\exists$-quantifier of first-order logic is a prime candidate but some other mechanisms like individual assertions $a : C$ in Description Logics[3] or the @ operator of hybrid logic[2] can be used.

Requirement 5 *$\mathcal{L}$ must be able to express $App(\rho[\boldsymbol{c}])$ for all rules $\rho[\boldsymbol{c}]$ of the graph rewrite system under study.*

Theorem 1 (Soundness). *Let $\mathcal{L}$ be a logic satisfying requirements 1 to 5. Let $SP = (Pre, Post, (\mathcal{R}, \mathcal{S}))$ be a specification. If $(Pre \Rightarrow wp(\mathcal{S}, Post)) \wedge vc(\mathcal{S}, Post)$ is valid in $\mathcal{L}$, then for all graphs G, G' such that $G \Rightarrow_{\mathcal{S}} G'$, $G \models Pre$ implies $G' \models Post$.*

Proof (Sketch). The proof of this theorem is quite straightforward. One just has to check for every atomic strategy s that if $Pre \Rightarrow wp(s, Post)$ and $G \models Pre$ then $G' \models Post$. We give the proof for the rule application which is the most complex.

Assume $\mathcal{S} = \rho[\boldsymbol{c}]$ where $\rho[\boldsymbol{c}]$ is a rule of $\mathcal{R}$. Let us assume $Pre \Rightarrow wp(\rho[\boldsymbol{c}], Post)$ is valid. Because $wp(\rho[\boldsymbol{c}], Post) = App(tag(\rho[\boldsymbol{c}])) \Rightarrow wp(tag(\alpha_{\rho[\boldsymbol{c}]}), Post)$, also $(Pre \wedge App(tag(\rho[\boldsymbol{c}]))) \Rightarrow wp(tag(\alpha_{\rho[\boldsymbol{c}]}), Post)$ is valid. Let G be a graph. If $G \models App(\rho[\boldsymbol{c}])$, there is a match h. Let G' be such that $G \Rightarrow_{\rho[\boldsymbol{c}],h} G'$. By definition of the substitutions, $G \Rightarrow_{\rho[\boldsymbol{c}],h} G'$ and $G \models wp(tag(\alpha_{\rho[\boldsymbol{c}]}), Post)$ implies $G' \models Post$. On the other hand, if $G \not\models App(\rho[\boldsymbol{c}])$, there does not exist any G' such that $G \Rightarrow_{\rho[\boldsymbol{c}]} G'$ and thus the program fails. Thus $G \models Pre$ implies that $G' \models Post$ $\square$.

After performing the calculus, one gets a formula $vc(\mathcal{S}, Post) \wedge (Pre \Rightarrow wp(\mathcal{S}, Post))$. Obviously, in order to be able to decide whether or not a program is correct, one has to prove that the obtained formula is valid. Hence the following requirement.

Requirement 6 *The validity problem for $\mathcal{L}$ is decidable.*

Nevertheless, this last requirement could be optional if interactive theorem provers are preferred.

5 Instances of the Example

Hereafter, we illustrate the general logical framework proposed in the previous section through the Hospital example by providing logics which fulfill the six proposed requirements. In [7] another instance is proposed using an extension of propositional dynamic logic is proposed.

First, let us observe that all of the invariants that we defined can be expressed in first-order logic (Formulae on the right).

Property 1:
$MS = NU \uplus PH$ $\rightsquigarrow \forall x.\ MS(x) \Leftrightarrow (NU(x) \wedge \neg\ PH(x)) \vee (\neg NU(x) \wedge\ PH(x))$

Property 2:
$PA \cup MS \subseteq PE$ $\rightsquigarrow \forall x. PA(x) \vee \mathrm{MS}(x) \Rightarrow PE(x)$

Property 3:
$write\_access \subseteq read\_access$ $\rightsquigarrow \forall x, y. write\_access(x, y) \Rightarrow read\_access(x, y)$

Property 4:
$read\_access \circ is\_about \subseteq treats$ $\rightsquigarrow \forall x, y, z. read\_access(x, y) \wedge is\_about(y, z) \Rightarrow treats(x, z)$

Property 5:
$treats \subseteq MS \times PA$ $\rightsquigarrow \forall x, y. treats(x, y) \Rightarrow MS(x) \wedge PA(y)$

Property 6:
$PA \Rightarrow \exists^{=1} referent\_phys$ $\rightsquigarrow \forall x. PA(x) \Rightarrow (\exists y.\ referent\_phys(x, y) \wedge \forall z. referent\_phys(x, z) \Rightarrow z = y)$

First-order logic is not decidable though, and thus one may want to use a different logic in order to be able to decide the correctness of the considered properties. In the following, we use the 2-variable fragment of first-order logic with counting ($\mathcal{C}^2$)[14] and $\exists^*\forall^*$, the fragment of first-order logic whose formula in prenex form are of the form $\exists i_0, \ldots, i_k. \forall j_0, \ldots, j_l. A(i_0, \ldots, i_k, j_0, \ldots, j_l)$.

In order to be able to distinguish between nodes of a model (active nodes) and those which are not part of a given model, we add to the signature of the logic a unary predicate *Active* which ranges over nodes and edges. Creating a new node becomes adding it to the *Active* nodes. This also requires to add that $\forall x, y. \neg Active(x) \Rightarrow (\bigwedge_{\psi \text{ an atomic unary predicate}} \neg\psi(x) \wedge \bigwedge_{r \text{ an atomic binary predicate}} \neg r(x, y) \wedge \neg r(y, x))$. I.e., non active nodes are not assumed to satisfy any property.

Let SPH be the specification $(Pre, Post, \mathcal{P})$ associated to the hospital example. Assume the strategy is $\mathcal{S} = New\_Ph[\text{NPH,NEONAT}]; Del\_Pa[\text{OPA}]$ while the considered rewrite system $\mathcal{R}$ is the one from Figure 4. This program $\mathcal{P}$ creates a new physician NPH and lets the patient OPA leave the hospital. Let *inv* denote

the conjunction of the expected properties. Let the precondition Pre be $inv \wedge \exists x.(\text{NEONAT}(x) \wedge \text{DE}(x)) \wedge \exists x.(\text{OPA}(x) \wedge \text{PA}(x)) \wedge \forall x.\neg\text{NPH}(x)$. Let the postcondition $Post$ be $inv \wedge \exists x, y.(\text{NPH}(x) \wedge \text{PH}(x) \wedge works\_in(x, y) \wedge \text{NEONAT}(y) \wedge DE(y))$. Proving the correctness of SPH amounts to proving that $Pre \Rightarrow wp(\mathcal{S}, Post)$ is valid. This is a formula in first-order logic. In the following two subsections, this specification is proven to be correct using two different decidable logics that are able to express parts of Pre and $Post$.

5.1 Two-Variable Logic with Counting : $\mathcal{C}^2$

$\mathcal{C}^2$ is the two-variable fragment of first-order logic with counting. Its formulas are those of first-order logic than can be expressed with only two variables and using the counting quantifier constructor $\exists^{<n}x.P$ expressing that there are less than n values x that satisfy P. In our case, this constructor will mostly be used to express that there exist less than n different r-successors of a given node.

Definition 9. *Let $\mathcal{U}$ be a set of unary predicates, $u \in \mathcal{U}$, $\mathcal{B}$ be a set of binary predicates, $b \in \mathcal{B}$, n an integer. A formula ϕ of $\mathcal{C}^2$ is defined as:*

$$\begin{aligned}
\phi &:= \top \mid \phi \wedge \phi \mid \neg\phi \mid \exists^{<n}x.\phi_x \mid \exists^{<n}y.\phi_y \\
\phi_x &:= \phi \mid u(x) \mid b(x,x) \mid \phi_x \wedge \phi_x \mid \neg\phi_x \mid \exists^{<n}x.\phi_x \mid \exists^{<n}y.\phi_{x,y} \\
\phi_y &:= \phi \mid u(y) \mid b(y,y) \mid \phi_y \wedge \phi_y \mid \neg\phi_y \mid \exists^{<n}y.\phi_y \mid \exists^{<n}x.\phi_{x,y} \\
\phi_{x,y} &:= \phi_x \mid \phi_y \mid b(x,y) \mid b(y,x) \mid \phi_{x,y} \wedge \phi_{x,y} \mid \neg\phi_{x,y} \mid \exists^{<n}x.\phi_{x,y} \mid \exists^{<n}y.\phi_{x,y}
\end{aligned}$$

As usual, $\bot$ means $\neg\top$, $\phi \vee \psi$ means $\neg(\neg\phi \wedge \neg\psi)$, $\phi \Rightarrow \psi$ means $\neg\phi \vee \psi$, $\exists^{\geq n}v.\phi$ means $\neg\exists^{<n}v.\phi$, $\exists v.\phi$ means $\exists^{\geq 1}v.\phi$, $\forall v.\phi$ means $\neg\exists v.\neg\phi$.

Definition 10. *Let $\mathcal{G} = (N, E, \mathcal{C}, \mathcal{R}, \phi_N, \phi_E, s, t)$ be a graph. We define the valuation of formulae as follows:*

$$\top^I = true$$

$$(\phi \wedge \psi)^I = \phi^I \text{ and } \psi^I$$

$$(\neg\phi)^I = \text{not } \phi^I$$

$$(\exists^{<n}x.\phi_x)^I = \begin{cases} true & \text{if there does not exist } n \text{ nodes } m_1, \ldots, m_n, \\ & \quad m_i \neq m_j \text{ for } 0 < i < j \leq n \text{ such that } m_i \models \phi_x \\ false & \text{otherwise} \end{cases}$$

$(\exists^{<n}y.\phi_y)^I$ is defined the same as $(\exists^{<n}x.\phi_x)^I$ but replacing x's with y's

Let us now focus on $m \models \phi_x$:

$$\begin{aligned}
m \models \phi &\quad \text{iff } \phi^I \\
m \models u(x) &\quad \text{iff } u \in \phi_N(m) \\
m \models b(x,x) &\quad \text{iff there exists } e \in E.s(e) = m,\ t(e) = m \text{ and } b = \phi_E(e) \\
m \models (\phi_x \wedge \psi_x) &\quad \text{iff } m \models \phi_x \text{ and } m \models \psi_x \\
m \models \neg\phi_x &\quad \text{iff } m \not\models \phi_x \\
m \models \exists^{<n}x.\phi_x &\quad \text{iff there does not exist } n \text{ nodes } m'_1, \ldots, m'_n, \\
&\qquad m_i \neq m_j \text{ for } 0 < i < j \leq n \text{ such that } m'_i \models \phi_x \\
m \models \exists^{<n}y.\phi_{x,y} &\quad \text{iff there does not exist } n \text{ nodes } w_1, \ldots, w_n, \\
&\qquad w_i \neq w_j \text{ for } 0 < i < j \leq n \text{ such that } (m, w_i) \models \phi_{x,y}
\end{aligned}$$

$m \models \phi_y$ is defined the same way but swapping the x's and the y's.

Let us now focus on $(m, m') \models \phi_{x,y}$*:*

$(m, m') \models \phi_x$ *iff* $m \models \phi_x$

$(m, m') \models \phi_y$ *iff* $m' \models \phi_y$

$(m, m') \models b(x, y)$ *iff there exists* $e \in E.s(e) = m$, $t(e) = m'$ *and* $b = \phi_E(e)$

$(m, m') \models b(y, x)$ *iff there exists* $e \in E.s(e) = m'$, $t(e) = m$ *and* $b = \phi_E(e)$

$(m, m') \models (\phi_{x,y} \wedge \psi_{x,y})$ *iff* $(m, m') \models \phi_{x,y}$ *and* $(m, m') \models \psi_{x,y}$

$(m, m') \models \neg\phi_{x,y}$ *iff* $(m, m') \not\models \phi_{x,y}$

$(m, m') \models \exists^{<n} x.\phi_{x,y}$ *iff there does not exist* n *nodes* $m_1, \ldots, m_n$,$m_i \neq m_j$ *for all* $0 < i < j \leq n$ *such that* $(m_i, m') \models \phi_{x,y}$

$(m, m') \models \exists^{<n} y.\phi_{x,y}$ *iff there does not exist* n *nodes* $m'_1, \ldots, m'_n$, $m'_i \neq m'_j$ *for all* $0 < i < j \leq n$ *such that* $(m, m'_i) \models \phi_{x,y}$

Theorem 2 ([14]). *The validity problem of* $\mathcal{C}^2$ *is decidable.*

Let us now check the six requirements of the previous section. $\mathcal{C}^2$ contains unary predicates that are interpreted on nodes and binary predicates that are interpreted on edges. *Pre* and *Post* are interpreted on graphs.

Theorem 3. $\mathcal{C}^2$ *is closed under substitutions.*

The proof relies on the fact that first-order logic is closed under substitution. The proof provides a system of rewrite rules that removes substitutions. As it does not introduce new variables, it also works for $\mathcal{C}^2$. We give three example rules to understand better how does it work:

- $(\phi \wedge \psi)[\sigma] \rightsquigarrow \phi[\sigma] \wedge \psi[\sigma]$ as if $\phi \wedge \psi$ is satisfied after performing σ, so must be ϕ and ψ and the other way round.
- $r(x, y)[r := r + (i, j)] \rightsquigarrow r(x, y) \vee (i(x) \wedge j(y))$ as $r^{I'}$ is $r^I \cup (i^I, j^I)$.
- $r(x, y)[clone(i, i')] \rightsquigarrow r(x, y) \vee (i'(x) \wedge \exists x.(i(x) \wedge r(x, y))) \vee (i'(y) \wedge \exists y.(i(y) \wedge r(x, y))) \vee (i'(x) \wedge i'(y) \wedge \exists x.(i(x) \wedge r(x, x)))$.

Example 2. $\mathcal{C}^2$ can express all the predicates $App(\rho)$ for the rules of the considered example (see Figure 4):

- $App(New\_Ph[\text{PH}_1,\text{D}_1]) = \exists x.(\text{D}_1(x) \wedge \text{DE}(x)) \wedge \exists x.(\neg Active(x) \wedge \text{PH}_1(x))$
- $App(New\_Pa[\text{PA}_1,\text{PH}_1,\text{FO}_1]) = \exists x, y.(\text{PH}_1(x) \wedge \text{PH}(x) \wedge works\_in(x, y)) \wedge \exists x.(\neg Active(x) \wedge \text{PA}_1(x)) \wedge \exists x.(\neg Active(x) \wedge \text{FO}_1(x))$
- $App(Del\_Pa[\text{PA}_1]) = \exists x, y.(\text{PA}_1(x) \wedge \text{PA}(x) \wedge hospitalized\_in(x, y))$
- $App(Del\_Ph[\text{PH}_1,\text{PH}_2]) = \exists x, y.(\text{PH}_1(x) \wedge \text{PH}(x) \wedge works\_in(x, y) \wedge \exists x.(\text{PH}_2(x) \wedge \text{PH}(x) \wedge works\_in(x, y)))$

One should also be interested in the ability of the logic to express the properties to be verified.

Example 3. $\mathcal{C}^2$ is not able to express Property 4: $read\_access \circ is\_about \subseteq treats$ as one would need to keep track of three variables at a time. On the other hand, Property 6: $\forall x.\text{PA}(x) \Rightarrow \exists^{=1} referent\_phys.\top$ is a formula of $\mathcal{C}^2$.

5.2 Exist-Forall-Prefix

The logic $\exists^*\forall^*$ is the fragment of first-order logic such that its prefix in prenex normal form is composed of a sequence of existential quantifiers and then a sequence of universal quantifiers.

Definition 11. *Let $\mathcal{U}$ be a set of unary predicates, $u \in \mathcal{U}$ and $\mathcal{B}$ a set of binary predicates, $b \in \mathcal{B}$. Let $x_1, \ldots, x_k, a_1, \ldots, a_l$ be variables and v, w denote two of them. A formula ϕ of $\exists^*\forall^*$ is defined as:*
$\phi := \exists x_0, \ldots, x_k, \forall a_0, \ldots, a_l.\psi(x_1, \ldots, x_k, a_1, \ldots, a_l)$
$\psi := \top \mid \psi \wedge \psi \mid \neg\phi \mid u(v) \mid b(v, w)$

As usual, $\bot$ means $\neg\top$, $\phi \vee \psi$ means $\neg(\neg\phi \wedge \neg\psi)$, $\phi \Rightarrow \psi$ means $\neg\phi \vee \psi$.

Definition 12. *Let $\mathcal{G} = (N, E, \mathcal{C}, \mathcal{R}, \phi_N, \phi_E, s, t)$ be a graph. We defined the valuation of formulae: $(\exists x_1, \ldots, x_k, \forall a_1, \ldots, a_l.\psi(x_0, \ldots, x_k, a_0, \ldots, a_l))^I = N$ iff there exist k nodes $(x_1, \ldots, x_k)$ such that for all choices of l nodes $(a_1, \ldots, a_l)$, $(x_1, \ldots, x_k, a_1, \ldots, a_l) \models \psi$.*
Let us define $(x_1, \ldots, x_k, a_1, \ldots, a_l) \models \psi$:
$(x_1, \ldots, a_l) \models \top$
$(x_1, \ldots, a_l) \models (\phi \wedge \psi)$ *iff* $(x_1, \ldots, a_l) \models \phi$ *and* $(x_1, \ldots, a_l) \models \psi$
$(x_1, \ldots, a_l) \models (\neg\phi)$ *iff* $(x_1, \ldots, a_l) \not\models \phi$
$(x_1, \ldots, a_l) \models u(v)$ *iff* $u \in \phi_N(v)$
$(x_1, \ldots, a_l) \models b(v, w)$ *iff there exists $e \in E$. $s(e) = v$, $t(e) = w$ and $b = \phi_E(e)$*

Theorem 4. *The validity problem of $\exists^*\forall^*$ is decidable.*

This is a well-known result ([8], chapter 6).

The six requirements of the previous section clearly hold for this logic. $\exists^*\forall^*$ contains unary predicates that are interpreted on nodes and binary predicates that are interpreted on edges.

Theorem 5. *$\exists^*\forall^*$ is closed under substitutions.*

The proof is exactly the same as the one for $\mathcal{C}^2$ and $\mathcal{FO}$. One needs to be careful though as additional quantifiers are introduced. They are always of the form $\exists x.(i(x) \wedge c(x))$ or $\exists x.(i(x) \wedge r(x, y))$ that can be rewritten as $\forall x.(\neg i(x) \vee c(x))$ or $\forall x.(\neg i(x) \vee r(x, y))$. Thus one can consider that only universal quantifiers are introduced.

Example 4. $\exists^*\forall^*$ can express all the predicates $App(\rho)$ for the rules of the considered example (see Figure 4):

- $App(New\_Ph[\mathrm{PH}_1, \mathrm{D}_1]) = \exists x.(\mathrm{D}_1(x) \wedge \mathrm{DE}(x)) \wedge \exists x.(\neg Active(x) \wedge \mathrm{PH}_1(x))$
- $App(New\_Pa[\mathrm{PA}_1, \mathrm{PH}_1, \mathrm{FO}_1]) = \exists x, y.(\mathrm{PH}_1(x) \wedge \mathrm{PH}(x) \wedge works\_in(x, y)) \wedge \exists x.(\neg Active(x) \wedge \mathrm{PA}_1(x)) \wedge \exists x.(\neg Active(x) \wedge \mathrm{FO}_1(x))$
- $App(Del\_Pa[\mathrm{PA}_1]) = \exists x, y.(\mathrm{PA}_1(x) \wedge \mathrm{PA}(x) \wedge hospitalized\_in(x, y))$
- $App(Del\_Ph[\mathrm{PH}_1, \mathrm{PH}_2]) = \exists x, y, z.(\mathrm{PH}_1(x) \wedge \mathrm{PH}(x) \wedge works\_in(x, y) \wedge \mathrm{PH}_2(z) \wedge \mathrm{PH}(z) \wedge works\_in(z, y))$

It is worth noting that the definition of $App(\rho)$ introduces new existential quantifiers as it checks for the existence of a match. This could seem to lead to a problem as the formula no longer is in $\exists^*\forall^*$. Actually, as the existentially quantified variables do not depend on the previously defined universally quantified variables, it is possible to move them at the beginning thus yielding a formula in $\exists^*\forall^*$.

Once more one has to check whether all properties can be expressed in the chosen logic.

Example 5. $\exists^*\forall^*$ is not able to express Property 6:$PA \Rightarrow \exists^{=1}$ *referent_phys* as it needs an existential quantifier after the universal ones to express the existence of an edge labeled with *referent_phys*. On the other hand, Property 4: $\forall x, y, z.read\_access(x, y) \wedge is\_about(y, z) \Rightarrow treats(x, z)$ is part of $\exists^*\forall^*$.

6 Conclusions

We considered the verification problem of model/graph transformations. We introduced a notion of specification consisting of pre- and postcondition which specify the correctness of the run of rewrite rules performed according to a given rewrite strategy.

Deciding the correctness of a given specification is not an easy and decidable task in general. We proposed some criteria which may be helpful to choose the most appropriate logics one can use to express proof obligations related to the correctness problem. We illustrated our proposal by considering a running example for which two decidable logics have been used to prove its correctness.

Even in the relatively simple considered example, none of the investigated logics is expressive enough to be able to deal with all the discussed properties. This is a deliberate choice. Our point is that one has to select for each problem one or several logics that are relevant and we proposed some criteria that help to select such logics.

References

1. S. Ahmetaj, D. Calvanese, M. Ortiz, and M. Simkus. Managing change in graph-structured data using description logics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 966–973, 2014.
2. C. Areces, P. Blackburn, and M. Marx. Hybrid logics: Characterization, interpolation and complexity. *J. Symb. Log.*, 66(3):977–1010, 2001.
3. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.
4. P. Balbiani, R. Echahed, and A. Herzig. A dynamic logic for termgraph rewriting. In *Procs. of ICGT 2010*, pages 59–74, 2010.
5. L. Baresi and P. Spoletini. *Procs. of ICGT 2006*, chapter On the Use of Alloy to Analyze Graph Transformation Systems, pages 306–320. Springer, 2006.

6. J. H. Brenas, R. Echahed, and M. Strecker. On the closure of description logics under substitutions. In *Proceedings of the 29th International Workshop on Description Logics, Cape Town, South Africa, April 22-25, 2016.*, 2016.
7. J. H. Brenas, R. Echahed, and M. Strecker. Proving correctness of logically decorated graph rewriting systems. In *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, pages 14:1–14:15, 2016.
8. E. Börger, E. Grädel, and Y. Gurevich. *The classical decision problem*. Springer, 2000.
9. A. Corradini, T. Heindel, F. Hermann, and B. König. Sesqui-pushout rewriting. In *Graph Transformations, ICGT 2006*, pages 30–45, 2006.
10. B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
11. L. M. de Moura and N. Bjørner. Z3: an efficient SMT solver. In *Procs. of TACAS 2008*, pages 337–340, 2008.
12. R. Echahed. Inductively sequential term-graph rewrite systems. In *4th International Conference on Graph Transformations, ICGT*, volume 5214 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2008.
13. A. H. Ghamarian, M. de Mol, A. Rensink, E. Zambon, and M. Zimakova. Modelling and analysis using GROOVE. *STTT*, 14(1):15–40, 2012.
14. E. Grädel, M. Otto, and E. Rosen. Two-Variable Logic with Counting is Decidable. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science LICS '97, Warschau*, 1997.
15. A. Habel and K. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(2):245–296, 2009.
16. S. Itzhaky, A. Banerjee, N. Immerman, A. Nanevski, and M. Sagiv. Effectively-propositional reasoning about reachability in linked data structures. In *Procs of CAV 2013*, pages 756–772, 2013.
17. D. Jackson. *Software Abstractions*. MIT Press, 2011.
18. K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In *Procs. of LPAR 2010*, pages 348–370. Springer, 2010.
19. R. Piskac, L. M. de Moura, and N. Bjørner. Deciding effectively propositional logic using DPLL and substitution sets. *J. Autom. Reasoning*, 44(4):401–424, 2010.
20. C. M. Poskitt and D. Plump. A hoare calculus for graph programs. In *Procs. of ICGT 2010*, pages 139–154, 2010.
21. C. M. Poskitt and D. Plump. Verifying monadic second-order properties of graph programs. In *Procs. of ICGT 2014*, pages 33–48, 2014.
22. J. C. Reynolds. An overview of separation logic. In *Verified Software: Theories, Tools, Experiments, First IFIP TC 2/WG 2.3 Conference, VSTTE 2005, Zurich, Switzerland, October 10-13, 2005, Revised Selected Papers and Discussions*, pages 460–469, 2005.
23. O. Semeráth, Á. Barta, Z. Szatmári, Á. Horváth, and D. Varró. Formal validation of domain-specific languages with derived features and well-formedness constraints. *International Journal on Software and Systems Modeling*, 07/2015 2015.
24. J. Tschannen, C. A. Furia, M. Nordio, and N. Polikarpova. *Procs. of TACAS 2015*, chapter AutoProof: Auto-Active Functional Verification of Object-Oriented Programs, pages 566–580. Springer, 2015.
25. D. Varró. Automated formal verification of visual modeling languages by model checking. *Software and System Modeling*, 3(2):85–113, 2004.