Verifying Graph Transformation Systems with Description Logics

Jon Haël Brenas[1], Rachid Echahed[2], and Martin Strecker[3][0000−0001−9953−9871]

[1] UTHSC - ORNL, Memphis, Tennessee, USA, jhael@uthsc.edu
[2] CNRS and University Grenoble-Alpes, LIG Lab. Grenoble, France, rachid.echahed@imag.fr
[3] Université de Toulouse, IRIT, Toulouse, France, martin.strecker@irit.fr

Abstract. We address the problem of verification of graph transformations featuring actions such as node *merging* and *cloning*, *addition* and *deletion* of nodes and edges, node or edge *labeling* and edge *redirection*. We introduce the considered graph rewrite systems following an algorithmic approach and then tackle their formal verification by proposing a Hoare-like weakest precondition calculus. Specifications are defined as triples of the form {Pre}(R, strategy){Post} where Pre and Post are conditions specified in a given Description Logic (DL), R is a graph rewrite system and strategy is an expression stating in which order the rules in R are to be performed. We prove that the proposed calculus is sound and characterize which DL logics are suited or not for the targeted verification tasks, according to their expressive power.

1 Introduction

Graphs, as well as their transformations, play a central role in modelling data in various areas such as chemistry, civil engineering or computer science. In many such applications, it may be desirable to be able to prove that graph transformations are correct, i.e., from any graph (or state) satisfying a given set of conditions, only graphs satisfying another set of conditions can be obtained.

The correctness of graph transformations has attracted some attention in recent years, see e.g., [24,16,21,6,5,19,3,9]. In this paper, we provide a Hoare-like calculus to address the problem of correctness of programs defined as strategy expressions over graph rewrite rules. Specifications are defined as triples of the form {Pre}(R, strategy){Post} where Pre and Post are conditions, R is a graph rewrite system and strategy is an expression stating how rules in R are to be performed. Our work is thus close to [9,16,21] but differs both on the class of the considered rewrite systems as well as on the logics used to specify Pre and Post conditions.

The considered rewrite rules follow an algorithmic approach where the left-hand sides are attributed graphs and the right-hand sides are sequences of elementary actions [14]. Among the considered actions, we quote node and edge *addition* or *deletion*, node and edge *labelling* and edge *redirection*, in addition

to node *merging* and *cloning*. To our knowledge, the present work is the first to consider the verification of graph transformations including node cloning.

Hoare-like calculi for the verification of graph transformations have already been proposed with different logics to express the pre- and post-conditions. Among the most prominent approaches figure nested conditions [16,21] that are explicitly created to describe graph properties. The considered graph rewrite transformations are based on the double pushout approach with linear spans which forbid actions such as node merging and node cloning.

Other logics might be good candidates to express graph properties which go beyond first-order definable properties such as monadic second-order logic [13,22] or the dynamic logic defined in [5]. These logics are 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 the use of decidable logics such as fragments of Description Logics (DLs) to specify graph properties [1,8]. DLs [4] are being used heavily in formal knowledge representation languages such as OWL [2]. In this paper we consider the case where `Pre` and `Post` conditions are expressed in DLs and show which Description Logic can be used or not for the targeted verification problems.

The use of decidable logics contributes to the design of a push-button technology that gives definite and precise answers to verification problems. Model-checking [24] provides an alternative to our approach with ready to use tools, such as Alloy [7,18] or GROOVE [15]. The main issue with those tools is that they are restricted to finding counter-examples, instead of a full verification, when the set of possible models is infinite (or too large to be checked in a timely manner), and thus provide only part of the solution. On the other hand, techniques based on abstract interpretation (such as [23]) are not guaranteed to give correct answers and have a risk of false positive or negative.

The paper is organized as follows. Section 2 provides the considered definitions of graphs and the elementary graph transformation actions. Section 3 recalls useful notions of Descriptions Logics. In Section 4, we define the considered graph rewrite systems and strategies. Section 5 provides a sound Hoare-like calculus and states which DLs can be used or not for the considered program verification problems. Section 6 concludes the paper. The missing proofs and definitions can be consulted in [10].

2 Preliminaries

We first define the notion of *decorated graphs* we consider in this paper.

Definition 1 (Decorated Graph). *Let $\mathcal{C}$ (resp. $\mathcal{R}$) be a set of node labels (resp. edge labels). A* decorated graph *G over a graph alphabet $(\mathcal{C}, \mathcal{R})$ is a tuple $(N, E, \Phi_N, \Phi_E, s, t)$ where N is a set of* nodes*, E is a set of* edges*, Φ_N is a* node labeling *function, $\Phi_N : N \to \mathcal{P}(\mathcal{C})$, Φ_E is an* edge labeling *function, $\Phi_E : E \to \mathcal{R}$, s is a* source function *$s : E \to N$ and t is a* target function *$t : E \to N$.*

Notice that nodes are decorated by means of subsets of $\mathcal{C}$ while edges are labeled with a single element in $\mathcal{R}$.

Graph transformation systems considered in this paper follow an algorithmic approach based on the notion of *elementary actions* introduced below.

Definition 2 (Elementary action, action). *Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An* elementary action*, say a, may be of the following forms:*

- *a* node addition $add_N(i)$ *(resp.* node deletion $del_N(i)$*) where i is a new node (resp. an existing node). It creates the node i. i has no incoming nor outgoing edge and it is not labeled (resp. it deletes i and all its incident edges).*
- *a* node label addition $add_C(i, c)$ *(resp.* node label deletion $del_C(i, c)$*) where i is a node and c is a label in $\mathcal{C}_0$. It adds the label c to (resp. removes the label c from) the labeling of node i.*
- *an* edge addition $add_E(e, i, j, r)$ *(resp.* edge deletion $del_E(e, i, j, r)$*) where e is an edge, i and j are nodes and r is an edge label in $\mathcal{R}_0$. It adds the edge e with label r between nodes i and j (resp. removes all edges with source i and target j with label r).*
- *a* global edge redirection $i \gg j$ *where i and j are nodes. It redirects all incoming edges of i towards j.*
- *a* merge action $mrg(i, j)$ *where i and j are nodes. This action merges the two nodes. It yields a new graph in which the first node i is labeled with the union of the labels of i and j and such that all incoming or outgoing edges of any of the two nodes are gathered.*
- *a* clone action $cl(i, j, L_{in}, L_{out}, L_{l\_in}, L_{l\_out}, L_{l\_loop})$ *where i and j are nodes and L_{in}, L_{out}, L_{l_in}, L_{l_out} and L_{l_loop} are subsets of $\mathcal{R}_0$. It clones a node i by creating a new node j and connects j to the rest of a host graph according to different information given in the parameters $L_{in}, L_{out}, L_{l\_in}, L_{l\_out}, L_{l\_loop}$ as specified further below.*

The result of performing an elementary action *a on a graph $G = (N^G, E^G, \Phi_N^G, \Phi_E^G, s^G, t^G)$, written $G[a]$, produces the graph $G' = (N^{G'}, E^{G'}, \Phi_N^{G'}, \Phi_E^{G'}, s^{G'}, t^{G'})$ as defined in Figure 1. A (composite)* 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 $G[\epsilon] = G$ where ϵ is the empty sequence.*

The elementary action $cl(i, j, L_{in}, L_{out}, L_{l\_in}, L_{l\_out}, L_{l\_loop})$ might be not easy to grasp at first sight. It thus deserves some explanations. Let node j be a clone of node i. What would be the incident edges of the clone j? Answering this question is not straightforward. There are indeed different possibilities to connect j to the neighborhood of i. Figure 2 illustrates such a problem where node q_1', a clone of node q_1, has indeed different possibilities to be connected to the other nodes. In order to provide a flexible clone action, the user may tune the way the edges connecting a clone are treated through the five parameters $L_{in}, L_{out}, L_{l\_in}, L_{l\_out}, L_{l\_loop}$. All these parameters are subsets of the set of edge labels $\mathcal{R}_0$ and are explained informally below:

- L_{in} (resp. L_{out}) indicates that every incoming (resp. outgoing) edge e of i, which is not a loop, and whose label is in L_{in} (resp. L_{out}) is cloned as a new edge e' such that $s(e') = s(e)$ and $t(e') = j$ (resp. $s(e') = j$ and $t(e') = t(e)$).
- L_{l_in} indicates that every self-loop e over i whose label is in L_{l_in} is cloned as a new edge e' with $s(e') = i$ and $t(e') = j$. (see the blue arrow in Fig. 2).
- L_{l_out} indicates that every self-loop e over i whose label is in L_{l_out} is cloned as a new edge e' with $s(e') = j$ and $t(e') = i$. (see the red arrow in Fig. 2).
- L_{l_loop} indicates that every self-loop e over i whose label is in L_{l_loop} is cloned as a new edge e' which is a self-loop over j, i.e, $s(e') = j$ and $t(e') = j$. (see the self-loop over node q'_1 in Fig. 2).

The semantics of the cloning action $cl(i, j, L_{in}, L_{out}, L_{l\_in}, L_{l\_out}, L_{l\_loop})$ as defined in Fig. 1 use some auxiliary pairwise disjoint sets representing the new edges that are created according to how the clone j should be connected to the neighborhood of node i. These sets of new edges are denoted $E'_{in}, E'_{out}, E'_{l\_in}, E'_{l\_out}$ and E'_{l_loop}. They are provided with the auxiliary bijective functions in, out, l_in, l_out and l_loop as specified below.

1. E'_{in} is in bijection through function in with the set $\{e \in E^G |\ t^G(e) = i \wedge s^G(e) \neq i \wedge \Phi^G_E(e) \in L_{in}\}$,
2. E'_{out} is in bijection through function out with the set $\{e \in E^G |\ s^G(e) = i \wedge t^G(e) \neq i \wedge \Phi^G_E(e) \in L_{out}\}$,
3. E'_{l_in} is in bijection through function l_in with the set $\{e \in E^G |\ s^G(e) = t^G(e) = i \wedge \Phi^G_E(e) \in L_{l\_in}\}$,
4. E'_{l_out} is in bijection through function l_out with the set $\{e \in E^G |\ s^G(e) = t^G(e) = i \wedge \Phi^G_E(e) \in L_{l\_out}\}$,
5. E'_{l_loop} is in bijection through function l_loop with the set $\{e \in E^G |\ s^G(e) = t^G(e) = i \wedge \Phi^G_E(e) \in L_{l\_loop}\}$).

Informally, the set E'_{in} contains a copy of every incoming edge e of node i (i.e., $t^G(e) = i$), which is not a self-loop (i.e., $s^G(e) \neq i$), and having a label in L_{in}, (i.e., $\Phi^G_E(e) \in L_{in}$). L_{in} is thus used to select which incoming edges of node i are cloned. The other sets E'_{out}, E'_{l_in}, E'_{l_out} and E'_{l_loop} are defined similarly.

Example 1. Let $\mathcal{A}$ be the graph of Fig. 2a, over an alphabet $(\mathcal{C}_0, \mathcal{R}_0)$ such that $\{a, b\} \subseteq \mathcal{R}_0$. Performing the action $cl(q_1, q'_1, \mathcal{R}_0, \mathcal{R}_0, X, Y, Z)$ yields the graph presented in Fig. 2b where the blue-plain (resp. red-dashed, purple-dotted) edge exists iff X (resp. Y, Z) contains the label $\{a\}$.

Readers familiar with algebraic approaches to graph transformation may recognize the cloning flexibility provided by the recent AGREE approach [11]. The parameters of the clone action reflect somehow the embedding morphisms of AGREE-rules. Cloning a node according to the approach of Sesquipushout [12] could be easily simulated by instantiating all the parameters by the full set of edge labels, i.e., $cl(i, j, \mathcal{R}_0, \mathcal{R}_0, \mathcal{R}_0, \mathcal{R}_0, \mathcal{R}_0)$.

If $\alpha = add_C(i,c)$ **then:**
$N^{G'} = N^G$,$E^{G'} = E^G$,
$\Phi_N^{G'}(n) = \begin{cases} \Phi_N^G(n) \cup \{c\} & \text{if } n = i \\ \Phi_N^G(n) & \text{if } n \neq i \end{cases}$
$\Phi_E^{G'} = \Phi_E^G$, $s^{G'} = s^G$, $t^{G'} = t^G$

If $\alpha = add_E(e,i,j,r)$ **then:**
$N^{G'} = N^G$, $\Phi_N^{G'} = \Phi_N^G$
$E^{G'} = E^G \cup \{e\}$

$\Phi_E^{G'}(e') = \begin{cases} r & \text{if } e' = e \\ \Phi_E^G(e') & \text{if } e' \neq e \end{cases}$
$s^{G'}(e') = s^G(e')$ if $e' \neq e, s^{G'}(e) = i$
$t^{G'}(e') = t^G(e')$ if $e' \neq e, t^{G'}(e) = j$

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

If $\alpha = del_N(i)$ **then:**
$N^{G'} = N^G \backslash \{i\}$
$E^{G'} = E^G \backslash \{e | s^G(e) = i \vee t^G(e) = i\}$
$\Phi_N^{G'}$ 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 $\alpha = i \gg j$ **then:**
$N^{G'} = N^G$, $E^{G'} = E^G$
$\Phi_N^{G'} = \Phi_N^G$, $\Phi_E^{G'} = \Phi_E^G$,$s^{G'} = s^G$
$t^{G'}(e) = \begin{cases} j & \text{if } t^G(e) = i \\ t^G(e) & \text{if } t^G(e) \neq i \end{cases}$

If $\alpha = mrg(i,j)$ **then:**
$N^{G'} = N^G \backslash \{j\}, E^{G'} = E^G, \Phi_E^{G'}(e) = \Phi_E^G(e)$
$\Phi_N^{G'}(n) = \begin{cases} \Phi_N^G(i) \cup \Phi_N^G(j) & \text{if } n = i \\ \Phi_N^G(n) & \text{otherwise} \end{cases}$
$s^{G'}(e) = \begin{cases} i & \text{if } s^G(e) = j \\ s^G(e) & \text{otherwise} \end{cases}$
$t^{G'}(e) = \begin{cases} i & \text{if } t^G(e) = j \\ t^G(e) & \text{otherwise} \end{cases}$

If $\alpha = del_C(i,c)$ **then:**
$N^{G'} = N^G$,$E^{G'} = E^G$,
$\Phi_N^{G'}(n) = \begin{cases} \Phi_N^G(n) \backslash \{c\} & \text{if } n = i \\ \Phi_N^G(n) & \text{if } n \neq i \end{cases}$
$\Phi_E^{G'} = \Phi_E^G$, $s^{G'} = s^G$, $t^{G'} = t^G$

If $\alpha = del_E(e,i,j,r)$ **then:**
$N^{G'} = N^G$, $\Phi_N^{G'} = \Phi_N^G$
$E^{G'} = E^G \backslash \{a \in E^G \mid$
$s^G(a) = i$, $t^G(a) = j$ and $\Phi_E^G(a) = r\}$
$\Phi_E^{G'}$ 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 $E^{G'}$

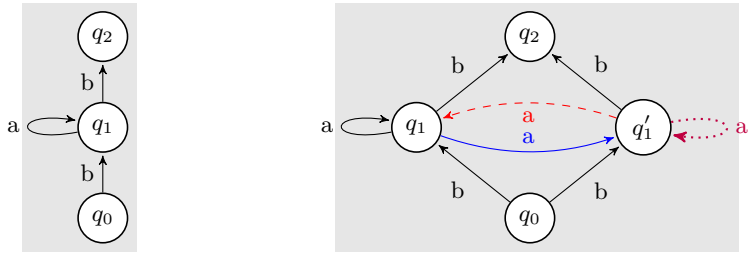
If $\alpha = cl(i,j,L_{in},L_{out},L_{l\_in},L_{l\_out},L_{l\_loop})$ **then:**
$N^{G'} = N^G \cup \{j\}$
$E^{G'} = E^G \cup E'_{in} \cup E'_{out} \cup E'_{l\_in} \cup E'_{l\_out} \cup E'_{l\_loop}$
$\Phi_N^{G'}(n) = \begin{cases} \Phi_N^G(i) & \text{if } n = j \\ \Phi_N^G(n) & \text{otherwise} \end{cases}$

$\Phi_E^{G'}(e) = \begin{cases} \Phi_E^G(in(e)) & \text{if } e \in E'_{in} \\ \Phi_E^G(out(e)) & \text{if } e \in E'_{out} \\ \Phi_E^G(l\_in(e)) & \text{if } e \in E'_{l\_in} \\ \Phi_E^G(l\_out(e)) & \text{if } e \in E'_{l\_out} \\ \Phi_E^G(l\_loop(e)) & \text{if } e \in E'_{l\_loop} \\ \Phi_E^G(e) & \text{otherwise} \end{cases}$

$s^{G'}(e) = \begin{cases} s^G(in(e)) & \text{if } e \in E'_{in} \\ j & \text{if } e \in E'_{out} \\ i & \text{if } e \in E'_{l\_in} \\ j & \text{if } e \in E'_{l\_out} \\ j & \text{if } e \in E'_{l\_loop} \\ s^G(e) & \text{otherwise} \end{cases}$

$t^{G'}(e) = \begin{cases} j & \text{if } e \in E'_{in} \\ t^G(out(e)) & \text{if } e \in E'_{out} \\ j & \text{if } e \in E'_{l\_in} \\ i & \text{if } e \in E'_{l\_out} \\ j & \text{if } e \in E'_{l\_loop} \\ t^G(e) & \text{otherwise} \end{cases}$

Fig. 1: $G' = G[\alpha]$, summary of the effects of the elementary actions: $add_N(i)$, $del_N(i)$, $add_C(i,c)$, $del_C(i,c)$, $add_E(e,i,j,r)$, $del_E(e)$, $i \gg j$, $mrg(i,j)$ and $cl(i,j,L_{in},L_{out},L_{l\_in},L_{l\_out},L_{l\_loop})$.



(a) A graph and (b) the possible results of cloning node q_1 as node q'_1

Fig. 2: Example of application of the elementary action *Clone*

3 DL logics

Description Logics (DLs) are a family of logic based knowledge representation formalisms. They constitute the basis of well known ontology languages such as the Web Ontology Language OWL [2]. Roughly speaking, a DL syntax allows one to define *Concept* names, which are equivalent to classical first-order logic unary predicates, *Role* names, which are equivalent to binary predicates and *Individuals*, which are equivalent to classical constants. There are various DLs in the literature, they mainly differ by the logical operators they offer to construct concept and role expressions or axioms. In this paper we are interested in extensions of the prototypical DL $\mathcal{ALC}$. We recall that these extensions are named by appending a letter representing additional constructors to the logic name. We focus on nominals (represented by $\mathcal{O}$), counting quantifiers ($\mathcal{Q}$), self-loops (*Self*), inverse roles ($\mathcal{I}$) and the universal role ($\mathcal{U}$). For instance, the logic $\mathcal{ALCUO}$ extends $\mathcal{ALC}$ with the universal role and nominals (see [4], for details about DL names). Below we recall the definitions of DL concepts and roles we consider in this paper.

Definition 3 (Concept, Role). *Let $\mathcal{A} = (\mathcal{C}_0, \mathcal{R}_0, \mathcal{O})$ be an alphabet where $\mathcal{C}_0$ (resp. $\mathcal{R}_0$ and $\mathcal{O}$) is a set of atomic concepts (resp. atomic roles and nominals). Let $c_0 \in \mathcal{C}_0$, $r_0 \in \mathcal{R}_0$, $o \in \mathcal{O}$, and n an integer. The set of concepts C and roles R are defined by:*

$C := \top \mid c_0 \mid \exists R.C \mid \neg C \mid C \vee C \mid o$ (*nominals*) $\mid \exists R.Self$ (*self loops*)
$\quad \mid (< \ n \ R \ C)$ (*counting quantifiers*)
$R := r_0 \mid U$ (*universal role*) $\mid R^-$ (*inverse role*)
For the sake of conciseness, we define $\bot \equiv \neg\top$, $C \wedge C' \equiv \neg(\neg C \vee \neg C')$, $\forall R.C \equiv \neg(\exists R.\neg C)$ *and* $(\geq \ n \ R \ C) \equiv \neg(< \ n \ R \ C)$.

The concepts C_{alc} and roles R_{alc} of the logic $\mathcal{ALC}$, which stand for "Attributive concept Language with Complement", are subsets of the concepts and roles given above. They can be defined as follows:

$C_{alc} := \top \mid c_0 \mid \exists R.C_{alc} \mid \neg C_{alc} \mid C_{alc} \vee C_{alc}$ and $R_{alc} := r_0$

Definition 4 (Interpretation). *An interpretation over an alphabet* $(\mathcal{C}_0, \mathcal{R}_0, \mathcal{O})$ *is a tuple* $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *where* $\cdot^{\mathcal{I}}$ *is a function such that* $c_0^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$*, for every atomic concept* $c_0 \in \mathcal{C}_0$*,* $r_0^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$*, for every atomic role* $r_0 \in \mathcal{R}_0$*,* $o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ *for every nominal* $o \in \mathcal{O}$*. The interpretation function is extended to concept and role descriptions by the following inductive definitions:*

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
- $(C \vee D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(\exists R.C)^{\mathcal{I}} = \{n \in \Delta^{\mathcal{I}} | \exists m, (n,m) \in R^{\mathcal{I}} \text{ and } m \in C^{\mathcal{I}}\}$
- $(\exists R.Self)^{\mathcal{I}} = \{n \in \Delta^{\mathcal{I}} | (n,n) \in R^{\mathcal{I}}\}$
- $(<\ n\ R\ C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} | \#(\{m \in \Delta^{\mathcal{I}} | (\delta, m) \in R^{\mathcal{I}} \text{ and } m \in C^{\mathcal{I}}\}) < n\}$
- $(R^-)^{\mathcal{I}} = \{(n,m) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} | (m,n) \in R^{\mathcal{I}}\}$
- $U^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

Definition 5 (Interpretation induced by a decorated graph). *Let* $G = (N, E, \Phi_N, \Phi_E, s, t)$ *be a graph over an alphabet* $(\mathcal{C}, \mathcal{R})$ *such that* $\mathcal{C}_0 \cup \mathcal{O} \subseteq \mathcal{C}$ *and* $\mathcal{R}_0 \subseteq \mathcal{R}$*. The interpretation induced by the graph* G*, denoted* $(\Delta^{\mathcal{G}}, \cdot^{\mathcal{G}})$*, is such that* $\Delta^{\mathcal{G}} = N$*,* $c_0^{\mathcal{G}} = \{n \in N | c_0 \in \Phi_N(n)\}$*, for every atomic concept* $c_0 \in \mathcal{C}_0$*,* $r_0^{\mathcal{G}} = \{(n,m) \in N \times N | \exists e \in E.s(e) = n \text{ and } t(e) = m \text{ and } r_0 = \Phi_E(e)\}$*, for every atomic role* $r_0 \in \mathcal{R}_0$*,* $o^{\mathcal{G}} = \{n \in N | o \in \Phi_N(n)\}$ *for every nominal* $o \in \mathcal{O}$*. We say that a node* n *of a graph* G *satisfies a concept* c*, written* $n \models c$ *if* $n \in c^{\mathcal{G}}$*. We say that a graph* G *satisfies a concept* c*, written* $G \models c$ *if* $c^{\mathcal{G}} = N$*, that is every node of* G *belongs to the interpretation of* c *induced by* G*. We say that a concept* c *is valid if for all graphs* G*,* $G \models c$*.*

To illustrate the different notions introduced in this paper, we use a running example inspired from ontologies related to the Malaria surveillance.

Example 2. Malaria is an infectious, vector-borne disease that overwhelmingly affects Sub-Saharan Africa. In order to reduce its incidence, one of the most widely used techniques is to install long lasting insecticide-treated nets (LLINs). Several materials can be used to produce LLINs and they can be treated with many different insecticides. Each insecticide has a mode of action that characterizes how it affects mosquitoes. In order to avoid the appearance of insecticide resistances in mosquito populations, it is required to use LLINs with different modes of actions.

In this example, we start by giving examples of concepts, roles and nominals related to Malaria surveillance. Let $\mathcal{A}_{mal} = (\mathcal{C}_{mal}, \mathcal{R}_{mal}, \mathcal{O}_{mal})$, be an alphabet such that $\{LLIN, Insecticide, Material, House, ModeOfAction\} \subseteq \mathcal{C}_{mal}$, $\{has\_ins, has\_mat, has\_moa, ins\_in\} \subseteq \mathcal{R}_{mal}$ and $\{i_0, l, mat, DDT\} \subseteq \mathcal{O}_{mal}$. We can then express as concepts that i_0 is an insecticide ($\exists U.i_o \wedge Insecticide$), that there exists a LLIN using the material mat ($\exists U.mat \wedge \exists has\_mat^-.LLIN$) or that all LLINs except for l are installed in at most one house ($\forall U.LLIN \Rightarrow ((<\ 2\ ins\_in\ House) \vee l)$).

4 Graph Rewrite Systems and Strategies

In this section, we introduce the notion of *DL decorated graph rewrite systems*. These are extensions of the graph rewrite systems defined in [14] featuring new actions over graph structures decorated over an alphabet $(\mathcal{C}, \mathcal{R})$ consisting of concepts $\mathcal{C}$ and roles $\mathcal{R}$ of a given DL logic.

Definition 6 (Rule, DLGRS). *A* rule *ρ is a pair (L, α) where L, called the left-hand side (lhs), is a decorated graph over $(\mathcal{C}, \mathcal{R})$ and α, called the right-hand side (rhs), is an action. Rules are usually written $L \to \alpha$. A DL decorated graph rewrite system, DLGRS, is a set of rules.*

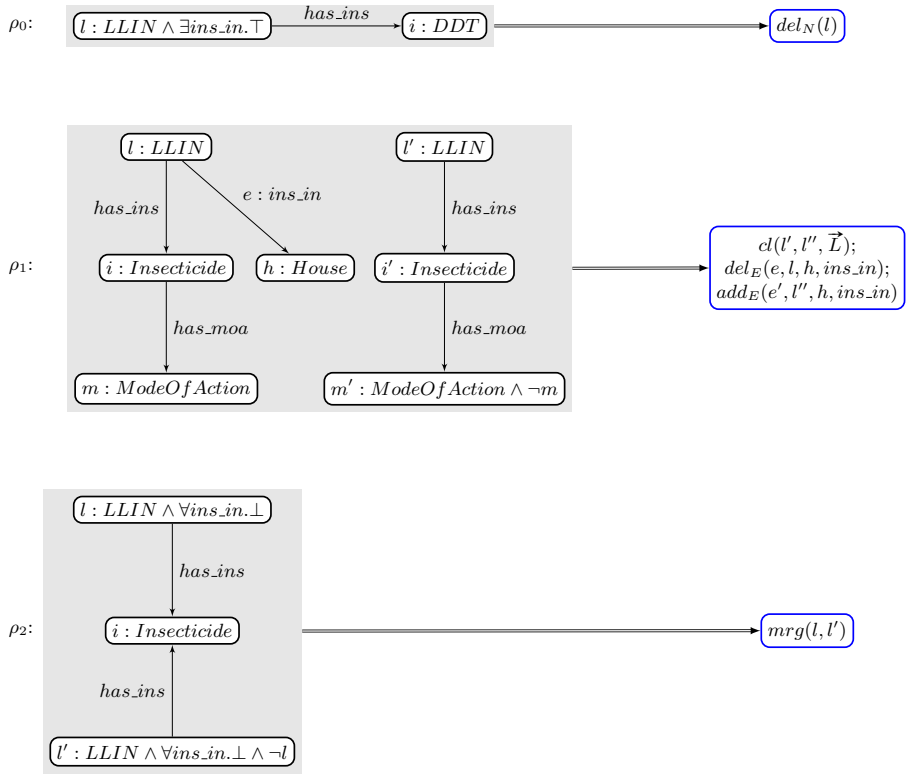


Fig. 3: Rules used in Example 3. In rule ρ_1, $\vec{\mathcal{L}} = \{\emptyset, \{has\_ins, has\_mat\}, \emptyset, \emptyset, \emptyset\}$.

Example 3. We now define some rules that can be applied to our malaria example. These rules are given in Fig. 3. Rule ρ_0 searches for a LLIN (l) using DDT

as an Insecticide and installed in a place, i.e., such that $l \models \exists ins\_in.\top$. As DDT is highly dangerous to human health, l is then deleted.

Rule ρ_1 changes the LLIN installed in a House. As aleady said in Example 2, one has to avoid using insecticides with the same mode of action twice in a row in the same House. ρ_1 thus searches for a LLIN (l) that is installed in a House (h) which has an Insecticide (i) with a given ModeOfAction (m). It also searches for a LLIN (l') that has an Insecticide (i') with a different ModeOfAction (m' where $m' \models \neg m$). A new LLIN (l'') is then created by cloning l', the edge, e, between l and h is removed (it loses its label) and a new one, e', is created between l'' and h labeled with ins_in.

The idea behind rule ρ_2 is that there are two kinds of LLINs: those that are currently used, i.e. installed in some House, and those that are used as templates for creating new LLINs by cloning with ρ_1. In order to limit the number of templates, if two LLINs which are not installed anywhere, i.e., they are models of $\forall ins\_in.\bot$, and they use the same Insecticide, then they can be considered as the same, and thus can be merged into one template by rule ρ_2.

Definition 7 (Match). *A* match *h between a lhs L and a graph G is a pair of functions $h = (h^N, h^E)$, with $h^N : N^L \to N^G$ and $h^E : E^L \to E^G$ such that:*
1. $\forall n \in N^L, \forall c \in \Phi^L_N(n), h^N(n) \models c$ *2.* $\forall e \in E^L, \Phi^G_E(h^E(e)) = \Phi^L_E(e)$
3. $\forall e \in E^L, s^G(h^E(e)) = h^N(s^L(e))$ *4.* $\forall e \in E^L, t^G(h^E(e)) = h^N(t^L(e))$

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 lhs, the node to which it is associated, $h(n)$, in G has to satisfy every concept in $\Phi^L_N(n)$. This condition clearly expresses additional negative and positive conditions which are added to the "structural" pattern matching. The second condition ensures that the match respects edge labeling.

Definition 8 (Rule application). *Let G be a graph decorated over an alphabet $(\mathcal{C}_0, \mathcal{R}_0)$ consisting of atomic concepts $\mathcal{C}_0$ and roles $\mathcal{R}_0$ of a given DL logic. G rewrites into graph G' using a rule $\rho = (L, \alpha)$ iff there exists a match h from L to G. G' is obtained from G by performing actions in $h(\alpha)$[4]. Formally, $G' = G[h(\alpha)]$. We write $G \to_\rho G'$.*

Very often, *strategies* are used to control the use of possible rules in rule-based programs (e.g. [26,20]). Informally, a strategy specifies the application order of different rules. It does not indicate where the matches are to be tried nor does it ensure unique normal forms.

Definition 9 (Strategy). *Given a graph rewrite system* $\mathtt{R}$*, a* strategy *is a word of the following language defined by s, where ρ is any rule in* $\mathtt{R}$*:*

$s := \epsilon$ *(Empty Strategy)* | ρ *(Rule)* | $s \oplus s$ *(Choice)*
$s; s$ *(Composition)* | s^* *(Closure)* | $\rho?$ *(Rule Trial)*
$\rho!$ *(Mandatory Rule)*

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

Informally, the strategy "$s_1; s_2$" means that strategy s_1 should be applied first, followed by the application of strategy s_2. The expression $s_1 \oplus s_2$ means that either the strategy s_1 or the strategy s_2 is applied. The strategy ρ^* means that rule ρ is applied as many times as possible. Notice that the closure is the standard "while" construct, that is the strategy s^* applies s as much as possible.

Example 4. Let us assume that we want to get rid of DDT-treated LLINs, change the LLINs in 1 or 2 Houses and then remove duplicate templates. In such a situation, one can use the strategy: $\rho_0^*; (\rho_1 \oplus (\rho_1; \rho_1)); \rho_2^*$.

We write $G \Rightarrow_{\mathtt{s}} G'$ to denote that graph G' is obtained from G by applying the strategy $\mathtt{s}$. In Fig. 4, we provide the rules that specify how strategies are used to rewrite a graph. For that we use the formula $\mathbf{App}(\mathtt{s})$ such that for all graphs G, $G \models \mathbf{App}(\mathtt{s})$ iff the strategy $\mathtt{s}$ can perform at least one step over G. This formula is specified below.

- $G \models \mathbf{App}(\rho)$ iff there exists a match h from the left-hand side of ρ to G
- $G \models \mathbf{App}(\rho!)$ iff there exists a match h from the left-hand side of ρ to G
- $G \models \mathbf{App}(\epsilon)$
- $G \models \mathbf{App}(s_0^*)$
- $G \models \mathbf{App}(\rho?)$
- $G \models \mathbf{App}(s_0 \oplus s_1)$ iff $G \models \mathbf{App}(s_0)$ or $G \models \mathbf{App}(s_1)$
- $G \models \mathbf{App}(s_0; s_1)$ iff $G \models \mathbf{App}(s_0)$

Whenever $G \models \mathbf{App}(s)$, this does not mean that the whole strategy, s, can be applied on G, but it rather ensures that at least one step of the considered strategy can be applied.

$$\frac{}{G \Rightarrow_{\epsilon} G} \text{ (Empty rule)} \qquad \frac{G \Rightarrow_{s_0} G'' \quad G'' \Rightarrow_{s_1} G'}{G \Rightarrow_{s_0;s_1} G'} \text{ (Strategy composition)}$$

$$\frac{G \Rightarrow_{s_0} G'}{G \Rightarrow_{s_0 \oplus s_1} G'} \text{ (Choice left)} \qquad \frac{G \Rightarrow_{s_1} G'}{G \Rightarrow_{s_0 \oplus s_1} G'} \text{ (Choice right)}$$

$$\frac{G \not\models \mathbf{App}(s)}{G \Rightarrow_{s^*} G} \text{ (Closure false)} \qquad \frac{G \Rightarrow_{s} G'' \quad G'' \Rightarrow_{s^*} G' \quad G \models \mathbf{App}(s)}{G \Rightarrow_{s^*} G'} \text{ (Closure true)}$$

$$\frac{G \not\models \mathbf{App}(\rho)}{G \Rightarrow_{\rho} \top} \text{ (Rule False)} \qquad \frac{G \models \mathbf{App}(\rho) \quad G \to_{\rho} G'}{G \Rightarrow_{\rho} G'} \text{ (Rule True)}$$

$$\frac{G \not\models \mathbf{App}(\rho)}{G \not\Rightarrow_{\rho!}} \text{ (Mandatory Rule False)} \qquad \frac{G \models \mathbf{App}(\rho) \quad G \to_{\rho} G'}{G \Rightarrow_{\rho!} G'} \text{ (Mandatory Rule True)}$$

$$\frac{G \not\models \mathbf{App}(\rho)}{G \Rightarrow_{\rho?} G} \text{ (Rule Trial False)} \qquad \frac{G \models \mathbf{App}(\rho) \quad G \to_{\rho} G'}{G \Rightarrow_{\rho?} G'} \text{ (Rule Trial True)}$$

Fig. 4: Strategy application rules

Notice that the three strategies using rules (i.e. ρ, $\rho!$ and $\rho?$) behave the same way when $G \models \mathbf{App}(\rho)$ holds, as shown in Figure 4, but they do differ when $G \not\models \mathbf{App}(\rho)$. In such a case, ρ can yield any graph, denoted by $\top$, (i.e. the process stops without an error), $\rho!$ stops the rewriting process with failure and $\rho?$ ignores the rule application and moves to the next step to be performed, if any, of the considered strategy.

The formula **App** has to be able to express the existence of a match in the considered logic. However, assuming the nodes of the lhs are explicitly named, $L = (\{n_0, \ldots, n_k\}, E, \Phi_N, \Phi_E, s, t)$, for a rule ρ, one may specify the existence of a match in a more direct way when using explicitly one nominal o_i for each node n_i of the lhs. That is to say, one can define a predicate $App(\rho, \{o_0, \ldots, o_k\})$, also noted $App(\rho)$, such that $G \models App(\rho, \{o_0, \ldots, o_k\})$ iff there exists a match h such that $h_N(n_0) = o_0^G, \ldots, h_N(n_k) = o_k^G$. This requires less expressive power to express than **App**. We also define $NApp(\rho) \equiv \neg\mathbf{App}(\rho)$.

Example 5. For the rule ρ_2 of Figure 3, $App(\rho_2, \{l_0, i_0, l_0'\}) \equiv \exists U.(l_0 \wedge LLIN \wedge \forall ins\_in.\bot \wedge \exists has\_ins.(i_0 \wedge Insecticide \wedge \exists has\_ins^-.(l_0' \wedge LLIN \wedge \neg l \wedge \forall ins\_in.\bot)))$.

5 Verification

In this section, we follow a Hoare style verification technique to specify properties of DLGRS's for which we establish a sound proof procedure.

Definition 10 (Specification). *A* specification *SP is a triple $\{\texttt{Pre}\}(\texttt{R}, \texttt{s})\{\texttt{Post}\}$ where $\texttt{Pre}$ and $\texttt{Post}$ are DL formulas, $\texttt{R}$ is a DLGRS and $\texttt{s}$ is a strategy.*

Example 6. Continuing with the malaria example, we give a simple example of a specification. We first define a precondition as *every LLIN is installed in at most one House*: $\texttt{Pre} \equiv \forall U.LLIN \Rightarrow (< 2\ ins\_in\ House)$. We can consider a post-condition having the same constraints as the precondition augmented with the fact that *no House is equipped with a LLIN using DDT*: $\texttt{Post} \equiv (\forall U.LLIN \Rightarrow (< 2\ ins\_in\ House)) \wedge (\forall U.House \Rightarrow \forall ins\_in^-.\forall has\_ins.\neg DDT)$. To complete the specification example, we consider the DLGRS given Fig. 3 and the strategy as proposed in Example 4.

Definition 11 (Correctness). *A specification SP is said to be* correct *iff for all graphs G, G' such that $G \Rightarrow_s G'$ and $G \models \texttt{Pre}$, then $G' \models \texttt{Post}$.*

In order to show the correctness of a specification, we follow a Hoare-calculus style [17] by computing weakest preconditions. For that, we give in Fig. 5 the definition of the function wp which yields the weakest precondition of a formula Q w.r.t. actions and strategies.

The weakest precondition of an elementary action, say a, and a postcondition Q is defined as $wp(a, Q) = Q[a]$ where $Q[a]$ stands for the precondition consisting of Q to which is applied a substitution induced by the action a that we denote by $[a]$. The notion of substitution used here follow the classical ones from Hoare-calculi (e.g., [25]).

Definition 12 (Substitutions). *A* substitution*, written* $[a]$*, is associated to each elementary action* a*, such that for all graphs* G *and DL formula* ϕ*,* $(G \models \phi[a]) \Leftrightarrow (G[a] \models \phi)$*.*

When writing a formula of the form $\phi[a]$, the substitution $[a]$ is used as a new formula constructor whose meaning is that the weakest preconditions for elementary actions, as defined above, are correct. DL logics are not endowed with such substitution constructor. The addition of such a substitution constructor to a given description logic is not harmless in general. That is to say, if ϕ is a formula of a DL logic $\mathcal{L}$, $\phi[a]$ is not necessarily a formula of $\mathcal{L}$. Hence, only closed DL logics under substitutions can be used for verification purposes. The two following theorems characterize non trivial fragments of DL logics which are closed, respectively not closed, under substitutions.

Theorem 1. *The description logics* $\mathcal{ALCUO}$, $\mathcal{ALCUOI}$, $\mathcal{ALCQUOI}$, $\mathcal{ALCUOS}elf$, $\mathcal{ALCUOIS}elf$*, and* $\mathcal{ALCQUOIS}elf$ *are closed under substitutions*

The proof of this theorem consists in providing a rewrite system which transforms any formula with substitutions into an equivalent substitution free formula in the considered logic. Details of such a rewrite system can be found in [10].

Theorem 2. *The description logics* $\mathcal{ALCQUO}$ *and* $\mathcal{ALCQUOS}elf$ *are not closed under substitutions.*

The proof of the above theorem is not straightforward. It uses notions of bisimulations induced by the considered logics. Two bisimilar models are provided which do not fulfill the same set of formulas. Details of the proof can be found in [10].

$wp(a,\ Q) = Q[a]$	$wp(a;\alpha,\ Q) = wp(a, wp(\alpha, Q))$
$wp(\epsilon,\ Q) = Q$	$wp(s_0; s_1,\ Q) = wp(s_0, wp(s_1,\ Q))$
$wp(s_0 \oplus s_1,\ Q) = wp(s_0, Q) \wedge wp(s_1, Q)$	$wp(s^*,\ Q) = inv_s$
$wp(\rho,\ Q) = App(\rho) \Rightarrow wp(\alpha_\rho, Q)$	$wp(\rho!, Q) = App(\rho) \wedge wp(\alpha_\rho, Q)$
$wp(\rho?, Q) = (App(\rho) \Rightarrow wp(\alpha_\rho, Q)) \wedge (\neg App(\rho) \Rightarrow Q)$	

Fig. 5: Weakest preconditions w.r.t. actions and strategies, where a (resp. α, α_ρ) stands for an elementary action (resp. action, the right-hand side of a rule ρ) and Q is a formula

In presence of DL logics closed under substitutions, the definitions of $wp(s, Q)$ for strategy expressions consisting of the *Empty Strategy*, the *Composition* or the *Choice* operators are quite direct (see, Fig. 5). The definitions of $wp(s, Q)$ when strategy s is a *Rule*, *Mandatory Rule* or *Rule Trial* are not the same depending on what happens if the considered rewrite rule cannot be applied. When a rule ρ

can be applied, then applying it should lead to a graph satisfying Q. When the rule ρ cannot be applied, $wp(\rho,\ Q)$ indicates that the considered specification is correct; while $wp(\rho!,\ Q)$ indicates that the specification is not correct and $wp(\rho?,\ Q)$ leaves the postcondition unchanged and thus transformations can move to possible next steps.

As for the computation of weakest preconditions for the *Closure* of strategies, it is close to the *while* statement. It requires an invariant inv_s to be defined, $wp(s^*,\ Q) = inv_s$, which means that the invariant has to be true when entering the iteration for the first time. On the other hand, it is obviously not enough to be sure that Q will be satisfied when exiting the iteration or that the invariant will be maintained throughout the execution. To make sure that iterations behave correctly, we need to introduce some additional *verification conditions* computed by means of a function vc, defined in Fig. 6.

$$
\begin{array}{ll}
vc(\epsilon,\ Q) & = vc(\rho,\ Q) = vc(\rho!, Q) = vc(\rho?, Q) = \top \text{ (true)} \\
vc(s_0; s_1,\ Q) & = vc(s_0, wp(s_1,\ Q)) \wedge vc(s_1, Q) \\
vc(s_0 \oplus s_1,\ Q) & = vc(s_0, Q) \wedge vc(s_1, Q) \\
vc(s^*,\ Q) & = vc(s, inv_s) \wedge (inv_s \wedge App(s) \Rightarrow wp(s, inv_s)) \wedge (inv_s \wedge \neg App(s) \Rightarrow Q)
\end{array}
$$

Fig. 6: Verification conditions for strategies.

As the computation of wp and vc requires the user to provide invariants, we now introduce the notion of annotated strategies and specification.

Definition 13 (Annotated strategy, Annotated specification). *An* annotated strategy *is a strategy in which every iteration s^* is annotated with an invariant inv_s. It is written $s^*\{inv_s\}$. An* annotated specification *is a specification whose strategy is an annotated strategy.*

Example 7. As the strategy introduced in Example 4 contains two closures, we need to define two invariants. We choose $inv_0 \equiv \texttt{Pre}$ and $inv_2 \equiv \texttt{Post}$. The annotated strategy we use is thus $\mathcal{AS} \equiv \rho_0^*\{inv_0\}; (\rho_1 \oplus (\rho_1; \rho_1)); \rho_2^*\{inv_2\}$.

Definition 14 (Correctness formula). *We call* correctness formula *of an annotated specification $SP = \{\texttt{Pre}\}(\mathcal{R}, \texttt{s})\{\texttt{Post}\}$, the formula : $correct(SP) = (\texttt{Pre} \Rightarrow wp(\texttt{s}, \texttt{Post})) \wedge vc(\texttt{s}, \texttt{Post})$.*

Example 8. The specification of the considered running example is thus $\{\texttt{Pre}\}(\mathcal{R}, \texttt{s})\{\texttt{Post}\}$, where `Pre` and `Post` are those introduced in Example 6, the rules $\mathcal{R}$ are those of Example 3 and the annotated strategy `s` is the strategy $\mathcal{AS}$ as defined in Example 7.

Theorem 3 (Soundness). *Let $SP = \{\texttt{Pre}\}(\mathcal{R}, \texttt{s})\{\texttt{Post}\}$ be an annotated specification. If $correct(SP)$ is valid, then for all graphs G, G' such that $G \Rightarrow_s G'$, $G \models \texttt{Pre}$ implies $G' \models \texttt{Post}$.*

The proof is done by structural induction on strategy expressions (see [10]).

Example 9. We now compute the correctness formula of the specification of Example 8: $corr \equiv (\texttt{Pre} \Rightarrow wp(\mathcal{AS}, \texttt{Post})) \wedge vc(\mathcal{AS}, \texttt{Post})$.

By applying the weakest precondition rules, $wp(\mathcal{AS}, \texttt{Post})) \equiv wp(\rho_0^*, wp((\rho_1 \oplus \rho_1; \rho_1); \rho_2^*, \texttt{Post})) \equiv inv_0$. Thus: $corr \equiv (\texttt{Pre} \Rightarrow inv_0) \wedge vc(\mathcal{AS}, \texttt{Post})$.

Let us now focus on $vc(\mathcal{AS}, \texttt{Post})$. For ease of reading, let us write $\mathcal{S}_1 \equiv (\rho_1 \oplus (\rho_1; \rho_1)); \rho_2^*\{inv_2\}$. By applying the rules for the verification conditions, one gets that $vc(\mathcal{AS}, \texttt{Post}) \equiv vc(\rho_0^*, wp(\mathcal{S}_1, \texttt{Post})) \wedge vc(\mathcal{S}_1, \texttt{Post})$. We will discuss the resulting subformulas.

1. We now focus on the first formula, $vc(\rho_0^*, wp(\mathcal{S}_1, \texttt{Post}))$. By applying the rules for the verification conditions, one gets that $vc(\rho_0^*, wp(\mathcal{S}_1, \texttt{Post})) \equiv vc(\rho_0, inv_0) \wedge (inv_0 \wedge App(\rho_0, \{l_0, i_0\}) \Rightarrow wp(\rho_0, inv_0)) \wedge (inv_0 \wedge NApp(\rho_0) \Rightarrow wp(\mathcal{S}_1, \texttt{Post}))$. The rules state that $vc(\rho, Q) = \top$; thus it is possible to get rid of the first formula $vc(\rho_0, inv_0)$.
   Similarly, $wp(\rho_0, inv_0) = App(\rho_0, \{l_0, i_0\}) \Rightarrow inv_0[del_N(l_0)]$.
   Altogether: $vc(\rho_0^*, wp(\mathcal{S}_1, \texttt{Post})) \equiv$
   $(inv_0 \wedge App(\rho_0, \{l_0, i_0\}) \Rightarrow inv_0[del_N(l_0)]) \wedge (inv_0 \wedge NApp(\rho_0) \Rightarrow wp(\mathcal{S}_1, \texttt{Post}))$.
   Let us now focus on the last wp. Unfolding the definition of $wp(\mathcal{S}_1, \texttt{Post}) \equiv wp(\rho_1 \oplus \rho_1; \rho_1, wp(\rho_2^*, \texttt{Post}))$. Applying the rules of weakest preconditions yields $wp(\rho_1, wp(\rho_2^*, \texttt{Post})) \wedge wp(\rho_1, wp(\rho_1, wp(\rho_2^*, \texttt{Post})))$. From the definition of the weakest precondition for a closure, one gets that $wp(\rho_2^*, \texttt{Post}) = inv_2$. By applying the rule for the weakest precondition of a rule application, one gets that:
   $wp(\rho_1, inv_2) \equiv App(\rho_1, \{l_1, h_1, i_1, m_1, l_1', i_1', m_1'\}) \Rightarrow inv_2\sigma_1$
   and
   $wp(\rho_1, wp(\rho_1, inv_2)) \equiv App(\rho_1, \{l_2, h_2, i_2, m_2, l_2', i_2', m_2'\}) \Rightarrow$
   $(App(\rho_1, \{l_3, h_3, i_3, m_3, l_3', i_3', m_3'\}) \Rightarrow inv_2\sigma_3)\sigma_2$
   where $\sigma_i \equiv [add_E(l_i'', h_i, ins\_in)][del_E(l_i'', h_i, ins\_in)][cl(l_i', l_i'', \vec{L})]$.
2. Let us now focus on the second formula. Let us apply the verification conditions rules, $vc(\mathcal{S}_1, \texttt{Post}) \equiv vc(\rho_1 \oplus \rho_1; \rho_1, wp(\rho_2^*, \texttt{Post})) \wedge vc(\rho_2^*, \texttt{Post})$. More applications of those rules yield $vc(\rho_1 \oplus \rho_1; \rho_1, wp(\rho_2^*, \texttt{Post})) \equiv \top$.
   Thus: $vc(\mathcal{S}_1, \texttt{Post}) \equiv vc(\rho_2^*, \texttt{Post})$
   We apply again the rule for closures to $vc(\rho_2^*, \texttt{Post})$ and get $vc(\rho_2, inv_2) \wedge (App(\rho_2, \{l_4, i_4, l_4'\}) \wedge inv_2 \Rightarrow wp(\rho_2, inv_2)) \wedge (inv_2 \wedge NApp(\rho_2) \Rightarrow \texttt{Post})$.
   Applying the verification rules yields $vc(\rho_2, inv_2) \equiv \top$.
   Applying the weakest precondition to $wp(\rho_2, inv_2)$ yields $App(\rho_2, \{l_4, i_4, l_4'\}) \Rightarrow inv_2[mrg(l_4, l_4')]$ and thus: $vc(\mathcal{S}_1, \texttt{Post}) \equiv$
   $(App(\rho_2, \{l_4, i_4, l_4'\}) \wedge inv_2 \Rightarrow inv_2[mrg(l_4, l_4')]) \wedge (inv_2 \wedge NApp(\rho_2) \Rightarrow \texttt{Post})$

To sum up, the correctness formula is $corr \equiv (\texttt{Pre} \Rightarrow inv_0) \wedge$
$(inv_0 \wedge App(\rho_0, \{l_0, i_0\}) \Rightarrow inv_0[del_N(l_0)]) \wedge$
$(inv_0 \wedge NApp(\rho_0) \wedge App(\rho_1, \{l_1, h_1, i_1, m_1, l_1', i_1', m_1'\}) \Rightarrow inv_2\sigma_1) \wedge$
$(inv_0 \wedge NApp(\rho_0) \wedge App(\rho_1, \{l_2, h_2, i_2, m_2, l_2', i_2', m_2'\}) \wedge$
$App(\rho_1, \{l_3, h_3, i_3, m_3, l_3', i_3', m_3'\})\sigma_2 \Rightarrow inv_1\sigma_3\sigma_2) \wedge$
$(inv_2 \wedge App(\rho_2, \{l_4, i_4, l_4'\}) \Rightarrow inv_2[mrg(l_4, l_4')]) \wedge (inv_2 \wedge NApp(\rho_2) \Rightarrow \texttt{Post})$.

The intermediate lines are the result of (1) and the last line is the result of (2). Furthermore,

- `Pre`, `Post`, inv_0 and inv_2 are defined in Example 6 and Example 7;
- $App(\rho_0, \{l_0, i_0\}) \equiv \exists U.(l_0 \wedge LLIN \wedge \exists ins\_in.\top \wedge \exists has\_ins.(i_0 \wedge DDT))$;
- $NApp(\rho_0) \equiv \forall U.(\neg LLIN \vee \forall ins\_in.\bot \vee \forall has\_ins.\neg DDT)$;
- $App(\rho_1, \{l_i, h_i, i_i, m_i, l'_i, i'_i, m'_i\}) \equiv \exists U.(l_i \wedge LLIN \wedge \exists ins\_in.(h_i \wedge House) \wedge \exists has\_ins.(i_i \wedge Insecticide \wedge \exists has\_moa.(m_i \wedge ModeOfAction))) \wedge \exists U.(l'_i \wedge \exists has\_ins.(i'_i \wedge \exists has\_moa.(m'_i \wedge ModeOfAction \wedge \neg m_i)))$;
- $App(\rho_2, \{l_4, i_4, l'_4\}) \equiv \exists U.(l_4 \wedge LLIN \wedge \forall ins\_in.\bot \wedge \exists has\_ins.(i_4 \wedge Insecticide \wedge \exists has\_ins^{-}.(l'_4 \wedge LLIN \wedge \forall ins\_in.\bot \wedge \neg l_4)))$ and
- $NApp(\rho_2) \equiv \forall U.Insecticide \Rightarrow (<\ 2\ has\_ins^{-}\ (LLIN \wedge \forall ins\_in.\bot))$.

6 Conclusion

We have presented a class of graph rewrite systems, DLGRSs, where the lhs's of the rules can express additional application conditions defined as DL logic formulas and rhs's are sequences of actions. The considered actions include node merging and cloning, node and edge addition and deletion among others. We defined computations with these systems by means of rewrite strategies. There is certainly much work to be done around such systems with logically decorated lhs's. For instance, the extension to narrowing derivations would use an involved unification algorithm taking into account the underlying DL logic. We have also presented a sound Hoare-like calculus and shown that the considered verification problem is still decidable with a large class of DL logics. Meanwhile, we pointed out two rich DL logics which are not closed under substitutions and thus cannot be candidate for verification issues. Future work includes an implementation of the proposed verification technique (work in progress) as well as the investigation of more expressive logics with connections to some SMT solvers.

References

1. S. Ahmetaj, D. Calvanese, M. Ortiz, and M. Simkus. Managing change in graph-structured data using description logics. In *Proc. of the 28th AAAI Conf. on Artificial Intelligence (AAAI 2014)*, pages 966–973. AAAI Press, 2014.
2. G. Antoniou, , G. Antoniou, G. Antoniou, F. V. Harmelen, and F. V. Harmelen. Web ontology language: Owl. In *Handbook on Ontologies in Information Systems*, pages 67–92. Springer, 2003.
3. M. Asztalos, L. Lengyel, and T. Levendovszky. Formal specification and analysis of functional properties of graph rewriting-based model transformation. *Software Testing, Verification and Reliability*, 23(5):405–435, 2013.
4. F. Baader. Description logic terminology. In *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 485–495. 2003.
5. P. Balbiani, R. Echahed, and A. Herzig. A dynamic logic for termgraph rewriting. In *5th International Conference on Graph Transformations (ICGT)*, volume 6372 of *LNCS*, pages 59–74. Springer, 2010.

6. P. Baldan, A. Corradini, and B. König. A framework for the verification of infinite-state graph transformation systems. *Inf. Comput.*, 206(7):869–907, 2008.
7. 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.
8. J. H. Brenas, R. Echahed, and M. Strecker. A Hoare-like calculus using the $SROIQ^{\sigma}$ logic on transformations of graphs. In J. Diaz, I. Lanese, and D. Sangiorgi, editors, *Theoretical Computer Science*, volume 8705 of *Lecture Notes in Computer Science*, pages 164–178. Springer Berlin Heidelberg, 2014.
9. 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*, pages 14:1–14:15, 2016.
10. J. H. Brenas, R. Echahed, and M. Strecker. On the verification of logically decorated graph transformations. *CoRR*, abs/1803.02776, 2018.
11. A. Corradini, D. Duval, R. Echahed, F. Prost, and L. Ribeiro. AGREE - algebraic graph rewriting with controlled embedding. In F. Parisi-Presicce and B. Westfechtel, editors, *Graph Transformation - 8th International Conference, ICGT 2015*, volume 9151 of *Lecture Notes in Computer Science*, pages 35–51. Springer, 2015.
12. A. Corradini, T. Heindel, F. Hermann, and B. König. Sesqui-pushout rewriting. In *ICGT 2006*, volume 4178 of *LNCS*, pages 30–45. Springer, 2006.
13. B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
14. 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.
15. A. H. Ghamarian, M. de Mol, A. Rensink, E. Zambon, and M. Zimakova. Modelling and analysis using GROOVE. *STTT*, 14(1):15–40, 2012.
16. 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.
17. C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.
18. D. Jackson. *Software Abstractions*. MIT Press, Feb. 2012.
19. B. König and J. Esparza. Verification of graph transformation systems with context-free specifications. In *ICGT 2010*, volume 6372 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 2010.
20. D. Plump. The graph programming language GP. In *Procs of Third International Conference on Algebraic Informatics, CAI 2009*, volume 5725 of *Lecture Notes in Computer Science*, pages 99–122. Springer, 2009.
21. C. M. Poskitt and D. Plump. A Hoare calculus for graph programs. In *Procs. of ICGT 2010*, pages 139–154, 2010.
22. C. M. Poskitt and D. Plump. Verifying monadic second-order properties of graph programs. In *Procs. of ICGT 2014*, pages 33–48, 2014.
23. A. Rensink and D. Distefano. Abstract graph transformation. *Electronic Notes in Theoretical Computer Science*, 157(1):39 – 59, 2006. Proceedings of the Third International Workshop on Software Verification and Validation (SVV 2005).
24. A. Rensink, Á. Schmidt, and D. Varró. Model checking graph transformations: A comparison of two approaches. In *ICGT 2004, Rome*, volume 3256 of *Lecture Notes in Computer Science*, pages 226–241. Springer, 2004.
25. R. Virga. Efficient substitution in Hoare logic expressions. *Electr. Notes Theor. Comput. Sci.*, 41(3):35–49, 2000.

26. E. Visser. Stratego: A language for program transformation based on rewriting strategies. In *Procs of the 12th International Conference on Rewriting Techniques and Applications, RTA 2001*, volume 2051 of *Lecture Notes in Computer Science*, pages 357–362. Springer, 2001.