Rule-level verification of graph transformations for invariants based on edges' transitive closure

Christian Percebois, Martin Strecker, Hanh Nhi Tran

Rule-Level Verification of Graph Transformations for Invariants Based on Edges' Transitive Closure*

Christian Percebois, Martin Strecker, and Hanh Nhi Tran

IRIT (Institut de Recherche en Informatique de Toulouse)
Université de Toulouse, France
{Christian.Percebois,Martin.Strecker,Hanh-Nhi.Tran}@irit.fr

Abstract. This paper develops methods to reason about graph transformation rules for proving the preservation of structural properties, especially global properties on reachability. We characterize a graph transformation rule with an applicability condition specifying the matching conditions of the rule on a host graph as well as the properties to be preserved during the transformation. Our previous work has demonstrated the possibility to reason about a graph transformation at rule-level with applicability conditions restricted to Boolean combinations of edge expressions. We now extend the approach to handle the applicability conditions containing transitive closure of edges, which implicitly refer to an unbounded number of nodes. We show how these can be internalized into a finite pattern graph in order to enable verification of global properties on paths instead of local properties on edges only.

Keywords: graph transformations, verification, formal methods, transitive closure, global property.

1 Introduction

Graph transformations have numerous applications in computer science. Many of them can be considered safety critical and therefore have to satisfy stringent correctness requirements. Verifying the correctness of a transformation involves formulating a property to be verified using a suitable logic and providing a method for proving the correctness of a given transformation. Although many efforts have been made to prove properties about transformation systems, like confluence or termination, there is only little work on ensuring the correctness of transformations. One challenge here is that many verification problems turn out to be hard to express, or worse, to be undecidable on graphs.

Two popular strategies can be used for verification of graph transformations: model checking and theorem proving. Works on model checking involve exploring the set of reachable states of a graph transformation system with respect to

* Part of this research has been supported by the *Climt (Categorical and Logical Methods in Model Transformation)* project (ANR-11-BS02-016).

a start graph to ensure that the required conditions holds. This technique is in particular possible if the graph is small enough but puts immediate limits on the state space of problems, potentially enormous, that can be explored by model checkers. In contrast to the model checking approach, the theorem proving approach reasons about constraints on states, not about instances of states. The search space of a theorem prover is thus typically infinite, whereas the search space of model checkers are usually finite (though large). One drawback of the theorem proving approach is that it is typically done interactively with advanced proof skills and not automatically, as in model checking.

The work described in this paper adopts the theorem proving approach to prove that a transformation rule is correct when applied to an arbitrary graph, provided certain applicability conditions are met. Our aim is to develop methods allowing to demonstrate the preservation of a given structural property during a graph transformation. A fundamental question underlying our approach is: is it possible to reason about a graph transformation by just taking into account the elements appearing in the rule itself, without having to consider others that might exist in the graph where the rule is applied?

This paper homogenizes and continues the strands developed by the authors in previous papers [17,18], in which we have shown that reasoning about a transformation applied to an arbitrary graph can be essentially reduced to reasoning about a bounded portion of the graph, namely the image of the transformation rule in the target graph. However, in this previous work, the verifiable properties are restricted to those that can be formulated by Boolean combinations of simple edge relations, encompassing properties that hold for vertices or constant-size vertex-neighborhoods. Consequently, the proposed solution could not handle global properties that hold for the graph as a whole, thus concern possibly an infinite number of edges which are outside of the rule, as acyclicity and connectivity. Generally, global properties must be expressed and verified at a global level [3]. The challenges here are how to express those global properties in the rule's applicability condition and how to reason about a possibly infinite number of nodes and edges with a finite number of computations. In this paper, we extend our approach to deal with global properties on paths in the rules' applicability conditions, especially connectivity and separation, thus allow verifying global properties at the rule-level.

First we introduce *transitive closure patterns* to express that the rule can be applied provided that two nodes are connected via the transitive closure of an edge relation. Then we show that this pattern, even though referring to a possibly unbounded number of nodes, can be reduced to a verification on simple edges, thus allows automation in this case. We also point out that sometimes one has to stipulate the non-existence of a connection in the underlying graph. Reasoning about these negative connectivity patterns turns out to be much more difficult, and we cannot give a complete calculus. We present however some reasoning patterns that allow a simplification in common situations.

We have used the interactive proof assistant Isabelle to model the transformations and to carry out the proofs described in this paper.

The rest of the paper is structured as follows: after a summary of our graph transformation representation in Section 2, we give some introductory examples in Section 3. Then in Section 4 we describe the background of our approach for rule-level verification. Section 5 presents the main contributions of this paper to handle the applicability conditions having transitive closure patterns in order reduce them to a finite case. In Section 6 we discuss some significant related work. Then we conclude with a perspective on future work.

2 Graphs and Graph Transformations

In order to facilitate the understanding of the rest of this article, we summarize our way of representing graphs and graph transformations.

In its simplest form, a graph *gr* is a datatype with two functions *nodes* (yielding the set of the nodes of the graph) and *edges* (yielding the set of edges of the graph). An edge is just an ordered pair of nodes. The node set of a graph is assumed to be finite (and, consequently, is the edge set).

A graph transformation rule *gt* is characterized by the following elements:

- **Transfo** gives the rule's name, followed by a list of *parameters* that designate nodes of the graph that the rule is applied to.
- **Appcond** specifies under which *applicability condition* the rule can be applied to a given graph. This condition, having as only free variables the rule's parameters, is a *path formula* whose structure will be defined later.
- **Action** describes which nodes and edges are to be deleted or added during the transformation.

Visually represented, a graph transformation rule consists of a left-hand side graph (LHS) and a right-hand side graph (RHS). The rule's LHS presents the rule's applicability condition and the rule's RHS presents the result of the rule's actions.

The specification of the transformation in Figure 1 is given as follows. The transformation *Refactoring* is applicable to three nodes c_1, c_2 and c_3. The applicability condition is that there is an edge between c_1 and c_3 (written as $\ll c_1, c_3 \gg$) and another between c_2 and c_3. The action is to delete the edge $\ll c_1, c_3 \gg$ and to add one between c_1 and c_2.

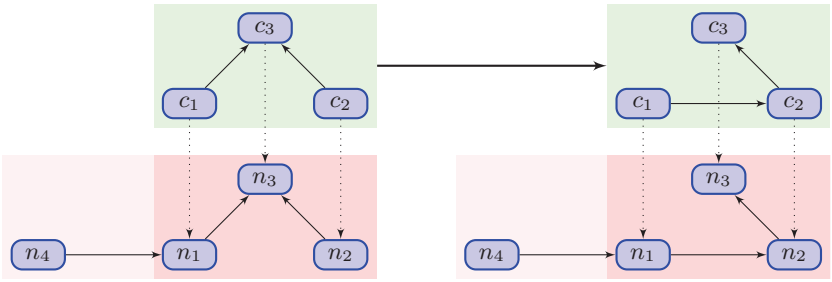

Fig. 1. Application of a graph transformation rule

Transfo *Refactoring*(c_1, c_2, c_3)
Appcond $\ll c_1, c_3 \gg \wedge \ll c_2, c_3 \gg$
Action *delete-edges:* $\ll c_1, c_3 \gg$
add-edges: $\ll c_1, c_2 \gg$

When applying the transformation rule gt to a host graph gr, we need the notion of *morphism* which maps the variables of the rule to nodes of the target graph. For example, in Figure 1 the morphism is the mapping $[c_1 \mapsto n_1, c_2 \mapsto n_2, c_3 \mapsto n_3]$. Quite naturally, some nodes of the graph gr might not be in the image of the morphism, e.g. node n_4.

For a transformation rule gt, its applicability condition *appcond* gt is represented by a path formula pf built on path expressions pe which are defined as follows:

$pe ::= \ll n_1, n_2 \gg$ - edge between nodes n_1 and n_2
$\mid\ n_1 \rightsquigarrow n_2$ - path between nodes n_1 and n_2
$pf ::= pe$ - elementary path formula
$\mid\ \neg\ pf$
$\mid\ pf \wedge pf$

Given a graph transformation gt, a host graph gr, a graph morphism gm:

- the predicate *path-form-interp* defines what it means for the applicability condition of gt, i.e. the path formula pf, to be satisfied under gm in gr.
- the application *apply-graphtrans-rel* performs the modifications specified in the action part of the transformation rule gt, by adding (respectively deleting) nodes and edges. The precise definition is technically more complex because it has to take deletion of dangling edges into account (c.f. [16]).

With these preliminaries, we can define *apply-transfo-rel*, the relation between a graph gr and the graph gr' resulting from applying the transformation gt to gr.

$$\frac{\exists gm.\textit{path-form-interp gr gm pf} \quad \wedge \quad \textit{apply-graphtrans-rel gt gr gr'}}{\textit{apply-transfo-rel gt gr gr'}}$$

This definition is entirely descriptive and not executable, because it imposes no choice as to which morphism gm (among several applicable morphisms) is selected. The graph gr' thus appears as a function of gr.

3 Illustrating Examples

To illustrate the motivations of our work, we use the example of refactoring navigation models to reorganize the set of web pages included in a web application and the links between those pages.

The rule displayed in Figure 2 describes a refactoring step that might be carried out on a navigation model. This rule refers explicitly to three pages c_1, c_2 and c_3. The solid arrow $\longrightarrow$ presents a direct navigation link r and the dashed arrow $- \rightarrow$ presents a navigation path r^* (reflexive-transitive closure of direct navigation links). The refactoring step consists in cutting the direct navigation link between c_1 and c_3 and introducing one between c_1 and c_2. As explained in

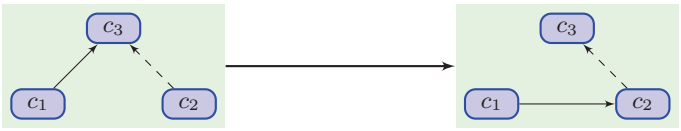

Fig. 2. Reorganizing a navigation model

Section 2, the rule's LHS presents the applicability condition. The application context might require that this refactoring only extends, but does not restrict the previous navigation possibility, for example in order to avoid that pages become unreachable. For example, the transformation keeps c_3 accessible from c_1 thanks to the navigation path $c_2 - \rightarrow c_3$. If r is the navigation relation before and r' the navigation relation after refactoring, we can express this preservation property more formally by the requirement $r^* \subseteq (r')^*$.

The delicate point about this transformation is the navigation path $c_2 - \rightarrow c_3$ in the rule's applicability condition, because it might be composed of some edges inside the rule, or might refer to an arbitrary number of intermediate nodes that are not explicitly mentioned in the rule. The next sections discuss three possible patterns of such a path and point out the problems to resolve for reducing the reasoning of graph transformation application to reasoning about the graph transformation rule.

3.1 Edge Conditions

The first example, displayed in Figure 1, describes the refactoring rule where the navigation path $c_2 - \rightarrow c_3$ contains just one edge (c_2, c_3). This example represents a more general case of transformation rules whose applicability condition includes only the elements described inside the rule. In other words, the applicability conditions of such rules can be expressed with path formulae that are essentially Boolean combinations of simple edge relations.

In our previous work [17], we have proposed a solution to reason locally about this kind of transformation rules to prove the preservation of reachability properties. We will briefly recapitulate the approach in Section 4.

3.2 Positive Path Conditions

Figure 3 shows the second example illustrating a more complicated case of applicability condition where the navigation path $c_2 - \rightarrow c_3$ is a reflexive-transitive closure of direct navigation links. Thus, this path might pass through nodes that are not described explicitly in the rule but appear in the host graph where the rule is applied. For example, when applying the rule on the graph in the lower part of Figure 3, we can see that the nodes in the image of the morphism (the dark-shaded area), namely n_2 and n_3, are connected by a path running through the outside node n_4.

Suppose that the transformation preserves the navigation path $c_2 - \rightarrow c_3$, this example typifies transformation rules requiring the existence of transitive closure

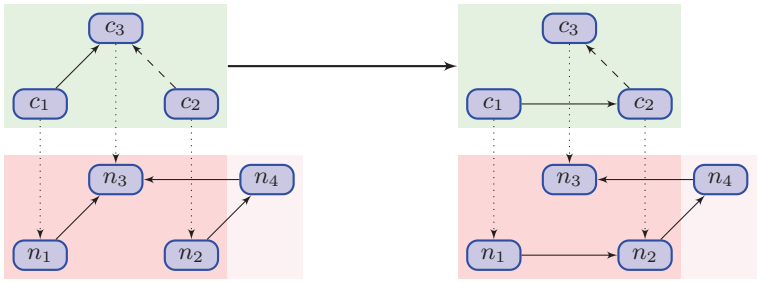

Fig. 3. Positive path condition ($n_2 \rightsquigarrow n_3$)

patterns in the rule's applicability condition. We call such conditions *"positive path conditions"*.

The first representation of the rule defined in Figure 2 is given in Figure 4a. The applicability condition, which is restricted to positive path condition in this example, is that there is an edge between c_1 and c_3 (written as $\ll c_1, c_3 \gg$) and a path between c_2 and c_3 (written as $c_2 \rightsquigarrow c_3$).

Transfo	$Refactoring(c_1, c_2, c_3)$
Appcond	$\ll c_1, c_3 \gg \wedge (c_2 \rightsquigarrow c_3)$
Action	*delete-edges:* $\ll c_1, c_3 \gg$
	add-edges: $\ll c_1, c_2 \gg$

(a) positive path condition

Transfo	$Refactoring(c_1, c_2, c_3)$
Appcond	$\ll c_1, c_3 \gg \wedge (c_2 \rightsquigarrow c_3)$
	$\wedge \neg (c_2 \rightsquigarrow c_1)$
Action	*delete-edges:* $\ll c_1, c_3 \gg$
	add-edges: $\ll c_1, c_2 \gg$

(b) positive and negative path conditions

Fig. 4. Definition of the rule *Refactoring*

The question here is how to reduce reasoning about the transitive closure relation r^* in the rule's LHS to reasoning about the direct edge relation r.

In Section 5.1, we will propose a solution to eliminate positive path conditions in order to enable a rule-level verification of reachability properties.

3.3 Negative Path Conditions

Unfortunately, defined as in Figure 2, an application of the rule might lead to an incorrect result if the navigation path $c_2 - \rightarrow c_3$ is affected by the transformation. Figure 5 shows an example of such situations. As one can see in this example, the path between the images of c_2 and c_3, namely the navigation path $n_2 - \rightarrow n_3$, runs through nodes n_4 and n_1. While n_4 is outside the image of the rule in the graph, n_1 and n_3 are inside the rule's image. The transformation deletes the image of edge (c_1, c_3) on the graph, i.e. the edge (n_1, n_3) and the navigation path $n_2 - \rightarrow n_3$ is not preserved. Consequently, n_1 is no more connected to n_3 after application of the rule, contrary to the intention of the rule.

Specifically for this example, a (very strict) solution is to forbid a path between c_2 and c_1: $\neg(c_2 - \rightarrow c_1)$. This solution introduces a *"negative path condition"*

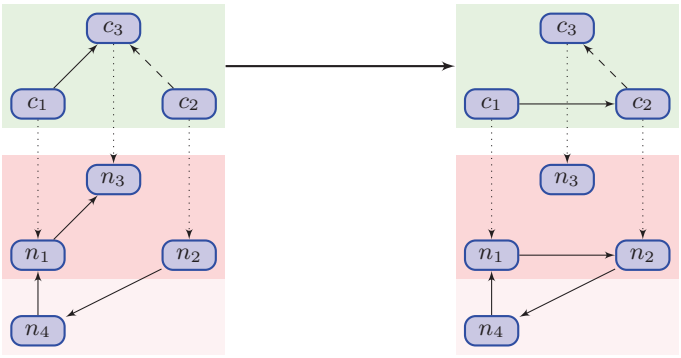

Fig. 5. Positive path condition ($n_2 \rightsquigarrow n_3$); Negative path condition $\neg(n_2 \rightsquigarrow n_1)$

into the rule's applicability condition besides the positive one and makes the local reasoning about the rule more difficult to deal with.

Figure 4b shows the rule in Figure 2 reinforced by forbidding the path between c_2 and c_1 to guarantee its correct applications on any graph. This second applicability contains then a negative path condition $\neg(c_2 \rightsquigarrow c_1)$.

In Section 5.1, we will outline where negative path conditions come into play during elimination of positive path conditions.

4 Rule-level Verification Based on Graph Decomposition

In this section, first we formalize the problem of graph transformation verification. Then, we briefly recapitulate our approach for reasoning about graph transformation at rule-level.

The properties that we want to prove are properties of global preservation of reachability or non-reachability (separation), formalized in the form

- $(edges\ gr)^* \subseteq (edges\ gr')^*$ for reachability or
- $(edges\ gr')^* \subseteq (edges\ gr)^*$ for non-reachability (separation).

where $edges\ gr$ is the edge relation of the original graph and $edges\ gr'$ is the edge relation of the transformed graph.

In [17], we have shown that if we restrict the applicability conditions to path formulae that are essentially Boolean combinations of simple edge relations, it is possible to reason about a graph transformation by just taking into account the nodes appearing in the rule itself, without having to consider other nodes that might exist in the graph where the rule is applied. We explain the main points of our approach in the following.

In a transformation, nodes included in a rule, denoted as node set A, represent free variables of the transformation's application condition, i.e. the possible images of the transformation under a given graph morphism. Embedding these conditions into a larger graph leads us to split the graph into an interior zone of A which is involved in the transformation, and an exterior zone of A which is a priori unaltered.

Since the properties of interest in this paper are mostly concerned with the edge relations of a graph, we define the *interior* and *exterior* of a relation r with respect to a node set A as follows:

definition $interior\ A\ r = r \cap (A \times A)$
definition $exterior\ A\ r = r \setminus (A \times A)$

So an edge belongs to the interior if both of its endpoints are in A; otherwise it belongs to the exterior.

An example is depicted in Figure 1: when choosing $A = \{n_1, n_2, n_3\}$ and $r = (edges\ gr)$, the interior zone is the dark-shaded part in Figure 1, i.e. $\{(n_1, n_3), (n_2, n_3)\}$, and the exterior zone is the light-shaded part, i.e. $\{(n_4, n_1)\}$.

The interior and exterior of a relation are disjoint and add up to the whole relation again:

$$interior\ A\ r \cap exterior\ A\ r = \emptyset \tag{1}$$
$$interior\ A\ r \cup exterior\ A\ r = r \tag{2}$$

Similar lemmas hold for transitive closure and reflexive-transitive closure; in the following we give those of reflexive-transitive closure:

$$((interior\ A\ r)^* \cap (exterior\ A\ r)^*)^* = \emptyset \tag{3}$$
$$((interior\ A\ r)^* \cup (exterior\ A\ r)^*)^* = r^* \tag{4}$$

The following two lemmas are at the heart of the decomposition method that we propose. When applied from right to left, they split up a goal into an exterior and an interior that can then be further simplified.

$$(interior\ A\ r \subseteq interior\ A\ s) \wedge (exterior\ A\ r \subseteq exterior\ A\ s) = (r \subseteq s) \tag{5}$$
$$((interior\ A\ r)^* \subseteq (interior\ A\ s)^*) \wedge ((exterior\ A\ r)^* \subseteq (exterior\ A\ s)^*) \Longrightarrow (r^* \subseteq s^*) \tag{6}$$

However, the converse of the last lemma does not hold in general. In fact, we will choose A to be the largest set of nodes i.e. nodes of the interior occurring in both relations r and s. We call $field$ this and so we have:

$$field\ s \subseteq A \wedge r^* \subseteq s^* \Longrightarrow (interior\ A\ r)^* \subseteq (interior\ A\ s)^* \tag{7}$$
$$field\ r \subseteq A \wedge r^* \subseteq s^* \Longrightarrow (exterior\ A\ r)^* \subseteq (exterior\ A\ s)^* \tag{8}$$

The detail proofs of the above lemmas are given in [16]. Examining the above subset containments of interior and exterior enables us to have a sound and also complete decomposition for a class of graphs where the region A has been chosen large enough.

Concretely, considering the edge relation *edges gr* (or variants with transitive closures $(edges\ gr)^*$) of a graph gr, to prove the preservation of reacha-

bility properties after the transformation on the graph gr', we replace the goal $(edges\ gr) \subseteq (edges\ gr')$ with two new goals:

$$(exterior\ A\ (edges\ gr)) \subseteq (exterior\ A\ (edges\ gr'))$$
$$\text{and } (interior\ A\ (edges\ gr)) \subseteq (interior\ A\ (edges\ gr'))$$

where in practice A will be the largest set of nodes whose existence is ascertained in the actual proof goal.

If the rules only have preconditions that are Boolean combinations of edge relations, it is sufficient to split the graph into an interior (the subgraph which lies entirely within the image of the rule's free variables under the graph morphism) and an exterior (the rest of the graph). The exterior of the graph can henceforth be disregarded; it is sufficient to verify the desired property on the interior of the graph, which can be done by a Boolean satisfiability check or, in the simplest case, by a symbolic computation.

Let us conclude this section by applying the above procedure to our example in Figure 1 to prove the preservation of all the paths in gr' after the transformation. The operational description of the rule specifies the addition of the edge (n_1, n_2) and the deletion of the edge (n_1, n_3) in the graph gr'. The injective morphism *injective-morphism* maps the variables of the rule to the nodes of the target graph gr. The initial goal is $(edges\ gr)^* \subseteq (edges\ gr')^*$. After expansion of definitions and some tidying of the proof state, this goal is derived into:

$$\begin{array}{l}[\![\ n_1 \in nodes\ gr; n_2 \in nodes\ gr; n_3 \in nodes\ gr;\\ \quad injective\text{-}morphism[c_1 \mapsto n_1, c_2 \mapsto n_2, c_3 \mapsto n_3];\\ \quad (n_1, n_3) \in edges\ gr; (n_2, n_3) \in edges\ gr;\\ \quad nodes\ gr' = nodes\ gr; edges\ gr' = \{(n_1, n_2)\} \cup (edges\ gr - \{(n_1, n_3)\})]\!]\\ \quad \Longrightarrow (edges\ gr)^* \subseteq (\{(n_1, n_2)\} \cup (edges\ gr - \{(n_1, n_3)\}))^*\end{array}$$

Since the variables n_1, n_2 and n_3 are free in our goal, we choose $A = \{n_1, n_2, n_3\}$, and obtain two new subgoals:

$$\begin{array}{l}(exterior\ \{n_1, n_2, n_3\}\ (edges\ gr))^*\\ \qquad \subseteq (exterior\ \{n_1, n_2, n_3\}\ (\{(n_1, n_2)\} \cup (edges\ gr - \{(n_1, n_3)\})))^*\\ \wedge\ (interior\ \{n_1, n_2, n_3\}\ (edges\ gr))^*\\ \qquad \subseteq (interior\ \{n_1, n_2, n_3\}\ (\{(n_1, n_2)\} \cup (edges\ gr - \{(n_1, n_3)\})))^*\end{array}$$

We develop the first goal by using the definition of exterior and the distributive property and get the refined goal:

$$\begin{array}{c}(exterior\ \{n_1, n_2, n_3\}\ (edges\ gr)) \subseteq (\{(n_1, n_2)\} \setminus (\{n_1, n_2, n_3\} \times \{n_1, n_2, n_3\})) \cup\\ (exterior\ \{n_1, n_2, n_3\}\ (edges\ gr) \setminus (\{(n_1, n_3)\} \setminus (\{n_1, n_2, n_3\} \times \{n_1, n_2, n_3\})))\end{array}$$

In this first goal, we have $(\{(n_1, n_2)\} \setminus (\{n_1, n_2, n_3\} \times \{n_1, n_2, n_3\})) = \emptyset$ and $(\{(n_1, n_3)\} \setminus (\{n_1, n_2, n_3\} \times \{n_1, n_2, n_3\})) = \emptyset$. Thus, the goal can be eliminated.

The second goal, after reductions by using the definition of interior, becomes

$\{(n_1, n_3), (n_2, n_3)\}^* \subseteq \{(n_1, n_2), (n_2, n_3)\}^*$ which can be verified by a simple symbolic computation.

5 Local Reasoning about Path Conditions

The question about local reasoning becomes more complex in the presence of transitive closures in the applicability conditions of a rule, as in the relation $c_2 - \rightarrow c_3$ in Figure 2. Dealing with transitive closure is a difficult problem that quickly becomes undecidable [8]. The existence of transitive closure patterns might be in the form of positive path conditions (as in Figure 4a) or negative path conditions (as in Figure 4b) that is considered as the adequate applicability condition of the rule in Figure 2:

$$\textbf{Appcond} \ll c_1, c_3 \gg \wedge (c_2 \rightsquigarrow c_3) \wedge \neg(c_2 \rightsquigarrow c_1)$$

This applicability condition requires the presence of the edge (c_1, c_3), the existence of a path between c_2 and c_3, but forbids a path between c_2 and c_1.

In the following, we outline:

- how to eliminate positive path conditions (Section 5.1);
- where negative path conditions come into play during elimination of positive paths (Section 5.1);
- after the reductions of transitive closure patterns, how to reason about graph transformations by applying the decomposition approach presented in Section 4 (Section 5.2).

We recall that we are mainly interested in problems of preservation of *reachability* of the form $(edges\ gr)^* \subseteq (edges\ gr')^*$. Slightly rewritten, this is the problem of showing $(x, y) \in (edges\ gr)^* \Rightarrow (x, y) \in (edges\ gr')^*$, for arbitrary x, y.

Lastly, the problems of preservation of *separation* are symmetric and can be handled with identical methods, so we only concentrate on the first kind of problem. To simplify the discussion and avoid complicated case distinctions, we furthermore make the assumption that graph morphisms are injective.

5.1 Materialization of Paths

The first step in our simplification procedure consists in replacing paths in our applicability conditions by edges in order to reduce reasoning about paths to reasoning about edges only. The following property justifies this step:

Lemma 1 (Path replacement)

$$(a, b) \in r^* \Longrightarrow (\{(a, b)\} \cup r)^* = r^*$$

and similarly for transitive closure $(.)^+$ *instead of reflexive-transitive closure* $(.)^*$.

The lemma expresses that a path $a \rightsquigarrow b$ known to exist in a graph can be materialized by adding the edge $\ll a, b \gg$ without changing the path relation.

Proof. In the following, we show the property for transitive closure only; reflexive-transitive closure is similar, but slightly more involved.

One direction of this equation is trivial by using monotonicity of the transitive closure relation:

$$r \subseteq r \cup \{(a,b)\} \Longrightarrow r^+ \subseteq (r \cup \{(a,b)\})^+$$

The direction $(\{(a,b)\} \cup r)^+ \subseteq r^+$ can be seen by expanding $(v,w) \in (\{(a,b)\} \cup r)^+$ into $(v,w) \in r^+ \vee ((v = a \vee (v,a) \in r^+) \wedge (b = w \vee (b,w) \in r^+))$ and then showing $(v,w) \in r^+$ by case distinction. These steps lead to four situations automatically resolved:

$[\![(a,b) \in r^+; v = a; b = w]\!] \Longrightarrow (v,w) \in r^+$,
$[\![(a,b) \in r^+; v = a; (b,w) \in r^+]\!] \Longrightarrow (v,w) \in r^+$,
$[\![(a,b) \in r^+; (v,a) \in r^+; b = w]\!] \Longrightarrow (v,w) \in r^+$,
$[\![(a,b) \in r^+; (v,a) \in r^+; (b,w) \in r^+]\!] \Longrightarrow (v,w) \in r^+$. □

In Lemma 1, we have dealt with the addition of a new edge; we need a related lemma for removal of an edge, as the edge $\ll c_1, c_3 \gg$ in the rule presented in Figure 4.

Lemma 2 (Deletion of unreachable edge)

$$(v,a) \notin r^* \Longrightarrow ((v,w) \in (r - \{(a,b)\})^* \Leftrightarrow ((v,w) \in r^*)$$

This lemma expresses that if a node a is not reachable from a node v in a graph, then any edge (a,b) starting from a can be removed without influencing the reachability from v.

Proof. The left to right direction is trivial: as $(r - \{(a,b)\}) \subseteq r$, so $(r - \{(a,b)\})^* \subseteq r^*$ by monotonicity of transitive closure.

To prove the other direction, we define the set $reach\ v\ r$ of nodes reachable from node v under relation r: $reach\ v\ r = \{w / (v,w) \in r^*\}$.

- The definition of $reach$ allows us to conclude :
$$(v,w) \in r^* \Longrightarrow (v,w) \in (r \cap (reach\ v\ r) \times (reach\ v\ r))^* \qquad \text{(a)}$$
- By assumption we have $(v,a) \notin r^*$ which means $a \notin reach\ v\ r$ and implies:
$$(a,b) \notin ((reach\ v\ r) \times (reach\ v\ r))^* \qquad \text{(b)}$$
- From (a) and (b) we can conclude that $(r \cap (reach\ v\ r) \times (reach\ v\ r))^* \subseteq (r - \{(a,b)\})^*$ and therefore $(v,w) \in (r - \{(a,b)\})^*$. □

5.2 Proving Preservation of Paths

Lemma 1 and Lemma 2 are used as conditional rewrite rules in the process of materialization. The starting point is to show that $(\{(a,b)\} \cup r)^* = r^*$, if there is a path $a \rightsquigarrow b$ in the applicability condition of a rule. During simplification, we may obtain subgoals of the form $(x,y) \in r^*$, which may be simplified by

- recursive use of Lemma 1,
- recursive use of Lemma 2,
- monotonicity rules of the form $(x,y) \in r^* \Longrightarrow (x,y) \in s^*$, for $r \subseteq s$.

To ensure termination, we do not try to simplify or to prove goals of the form $(x, y) \notin r^*$. Rather, these negative path conditions have to be directly given as hypotheses.

$$(x, y) \in (edges\ gr)^* \Longrightarrow (x, y) \in (edges\ gr')^*$$

where $edges\ gr$ is the edge relation of the original graph and $edges\ gr'$ is the edge relation of the transformed graph, possibly after addition of some edges that materialize paths.

To get rid of the abstract set $edges\ gr$ and $edges\ gr'$, we perform the graph decomposition presented in Section 4. By choosing A is the set of all nodes found in the transformation rule, $(interior\ A\ (edges\ gr))^*$ is the image of the rule LHS and $(exterior\ A\ (edges\ gr))^* = \emptyset$. Consequently, this process leaves us with the only goal:

$$(interior A\ (edges\ gr))^* \subseteq (interior A\ (edges\ gr'))^*$$

We can therefore reduce the global reasoning on gr and gr' to the local reasoning on the images of LHS and RHS.

With these observations, we can verify the transformation in our example in Figure 4b. In this example, we have the preconditions: $(x, y) \in (edges\ gr)^*$; $(n_1, n_3) \in edges\ gr$ and $(n_2, n_3) \in (edges\ gr)^*$. We furthermore have $(n_2, n_1) \notin (edges\ gr)^*$. Under these preconditions, we have to show

$$(x, y) \in (\{(n_1, n_2)\} \cup (edges\ gr - \{(n_1, n_3)\}))^*$$

We now materialized the path $n_2 \rightsquigarrow n_3$ by the edge (n_2, n_3), then showing that this goal is equivalent to

$$((x, y) \in (\{(n_2, n_3), (n_1, n_2)\} \cup (edges\ gr - \{(n_1, n_3)\}))^*$$

Proof. Indeed,

- By assumption we have $(n_2, n_1) \notin (edges\ gr)^*$. By Lemma 2 we can write:

$$(n_2, n_3) \in (edges\ gr - \{(n_1, n_3)\})^* \Leftrightarrow (n_2, n_3) \in (edges\ gr)^* \qquad \text{(a)}$$

- Then by monotonicity of reflexive-transitive closure we have:

$$\begin{aligned}(n_2, n_3) \in (edges\ gr - \{(n_1, n_3)\})^* \Longrightarrow \\ (n_2, n_3) \in (\{(n_1, n_2)\} \cup (edges\ gr - \{(n_1, n_3)\}))^*\end{aligned} \qquad \text{(b)}$$

- From (b) and using Lemma 1 to add the edge (n_2, n_3) to the set in (b), we have $(n_2, n_3) \in (\{(n_1, n_2)\} \cup (edges\ gr - \{(n_1, n_3)\}))^* \Longrightarrow$

$$\begin{aligned}(\{(n_2, n_3), (n_1, n_2)\} \cup (edges\ gr - \{(n_1, n_3)\}))^* \\ = (\{(n_1, n_2)\} \cup (edges\ gr - \{(n_1, n_3)\}))^*\end{aligned} \qquad \text{(c)}$$

Thanks to (c), we have the new equivalent proof goal $((x, y) \in (\{(n_2, n_3), (n_1, n_2)\} \cup (edges\ gr - \{(n_1, n_3)\}))^*$ which can then be tackled with the methods of Section 4.

Choosing $A = \{n_1, n_2, n_3\}$, after decomposing the graph, we get the goal

$$\{(n_1, n_3)\}^* \subseteq \{(n_2, n_3), (n_1, n_2)\}^*$$

which can be verified by a simple symbolic computation. □

6 Related Work

There are two main approaches in formal verification of graph transformations: solutions based on category theory and solutions based on a logical framework. The first one uses an underlying algebraic formalism as a framework for specifying and executing transformations. The second one defines a suitable logical framework to encode graphs and their properties, then uses inference methods to verify the properties on graphs as logical structures. While the category approach can propose efficient solutions, its level of generalization is rather low. Logical frameworks present general solutions but have to dealt with the problem of decidability and computational complexity. Some recent work on verification of graph transformations tries to take advantage of both of the above approaches.

In this paper we have followed the logical approach, however without defining a new logic, and focus on the verification of global properties. In the same spirit, Basil Becker et al. [1,2] proposed an automatic verification of invariants by creating symbolic representations for possible violations of the rule's properties. Then every transformation rule is inspected with respect to well-formedness constraints expressed either as a forbidden or a conditional forbidden pattern of the modeling language's meta-model. This work encodes graph patterns as first-order predicates, therefore it has to define additional maintenance rules in order to ensure global properties which cannot be expressed by forbidden or conditional forbidden patterns. In comparison with [2], we try to encapsulate global properties at the rule level by replacing paths with edges (see Lemma 1 and Lemma 2) without adding extra-predicates on nodes and edges which have to be analyzed during the verification process.

In [3] the authors analyzed global graph properties as connectivity, acyclicity and the Eulerian and Hamiltonian properties which are not definable in a basic modal logic. Then they proposed using a basic hybrid logic for some of these properties, a hybrid logic with a specific operator for Hamiltonian property and a hybrid logic together with a graded modal logic in order to handle numerical conditions for Eulerian property.

The work in [12] adds proposition graphs to transformation rules in order to compactly described feature connectivity patterns required during the transformation. The invariants to be verified are expressed in Computation Tree Logic (CTL). The main result of this paper states a satisfaction condition theorem for a transformation rule which preserves a given property. Close to us, [10] introduced the *-labelled edge notation as a replacement for a set of paths, each representing a possible sequence of edges. On the opposite, forbidden paths using regular expressions is proposed in the tool Augur2 [11].

In [9], the authors verified graph transformations written in Core UnCAL against the specified input/output graph structural constraints (schemes) in Monadic Second-Order logic (MSO). They first represented both Core UnCAL transformations and schemes by MSO formulas and then developed an algorithm to reduce the graph transformation verification problem to the validity of MSO over trees. The efficiency of this work relies on the algorithm to map the type-annotated Core UnCAL to a MSO-definable graph transduction, in conjunction with the decision procedure to verify MSO formulas.

The traditional algebraic approach has been also explored for reasoning on graph transformations. In this context, graph structures and properties are logically interpreted [13,5]. In [14,6] Pennemann et al. introduced the notion of nested graph conditions to describe structural properties. Since these conditions are first-order logic on graphs with a graphical representation of the nodes and edges, they cannot describe non-local graph properties. This approach extracts graph conditions and feeds them into SAT solvers or first-order theorem provers. However, there is no tight coupling between the semantics (expressed in categorical terms) and the proof obligation generator, and thus there is a dependency on a larger trusted code base.

In [7], the authors generalized the concept of nested graph conditions to Hyperedge Replacement conditions (HR) as conditions over graphs with variables. HR+ [15], the extension of HR, have been proposed as counterpart to MSO formulas to deal with global properties. The authors investigated the expressiveness of HR+ conditions and show that graphs with variables and replacement morphisms form a weak adhesive HLR category. Their conditions allow to express non-local properties of graphs.

In [4] the authors generalized Courcelle's notion of recognizable graph languages. They defined the logic on subobjects together with a procedure for translating MSO graph formulas into automaton functors for a class of categories including the category of graphs. This work allows defining complex properties such as "a subgraph is closed under reachability", or "there exists a path from x to y". However, this theoretical approach has practical consequences: graph decomposition into smaller units leads to a complex translation of graph formulas and more troublesome is the explosion of the state sets of automata which is still an open problem.

7 Conclusion

Expressive transformation patterns (such as transitive closure) that go beyond what is commonly used in graph rewriting systems are useful in some application domains, and they are amenable to a formal analysis. In this sense, we have presented simplification strategies that reduce reasoning about paths to reasoning about edges. These strategies can be understood as preprocessing steps carried out before verification procedures applicable to more restricted graph transformations.

The simplification method we have presented is sound, but not complete. Also, our approach is currently geared towards the preservation of particular properties

(reachability and separation). However, for dealing with the challenge of transitive closures, we think that our heuristic approach is a good compromise that we try to extend to other common reasoning patterns. We will also investigate more systematic sound and complete procedures, but for weaker logics.

As witnessed by our examples, it is difficult to get rules right; in particular, this means that some preconditions covering unsuspected special cases are usually missing. Another interesting line of research is therefore to help developers of rules find the right applicability patterns for transformations that are supposed to satisfy particular correctness conditions.

References

1. Becker, B., Beyer, D., Giese, H., Klein, F., Schilling, D.: Symbolic invariant verification for systems with dynamic structural adaptation. In: Proceedings of the 28th International Conference on Software Engineering, ICSE 2006 pp. 72–81. ACM (2006)
2. Becker, B., Lambers, L., Dyck, J., Birth, S., Giese, H.: Iterative development of consistency-preserving rule-based refactorings. In: Cabot, J., Visser, E. (eds.) ICMT 2011. LNCS, vol. 6707, pp. 123–137. Springer, Heidelberg (2011)
3. Benevides, M.R.F., Menasché Schechter, L.: Using modal logics to express and check global graph properties. Logic Journal of IGPL 17(5), 559–587 (2009)
4. Sander Bruggink, H.J., König, B.: A logic on subobjects and recognizability. In: Calude, C.S., Sassone, V. (eds.) TCS 2010. IFIP Advances in Information and Communication Technology, vol. 323, pp. 197–212. Springer, Heidelberg (2010)
5. Ehrig, H., Golas, U., Habel, A., Lambers, L., Orejas, F.: M-adhesive transformation systems with nested application conditions. Part 2: Embedding, critical pairs and local confluence. Fundam. Inf. 118(1-2), 35–63 (2012)
6. Habel, A., Pennemann, K.-H.: Correctness of high-level transformation systems relative to nested conditions. Mathematical Structures in Computer Science 19(02), 245–296 (2009)
7. Habel, A., Radke, H.: Expressiveness of graph conditions with variables. In: Electronic Communications of the EASST (2010)
8. Immerman, N., Rabinovich, A., Reps, T., Sagiv, M., Yorsh, G.: The boundary between decidability and undecidability for transitive-closure logics. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 160–174. Springer, Heidelberg (2004)
9. Inaba, K., Hidaka, S., Hu, Z., Kato, H., Nakano, K.: Graph-transformation verification using monadic second-order logic. In: Proceeding of the 13th International ACM SIGPLAN Symposium on Symposium on Principles and Practice of Declarative Programming. ACM Press (July 2011)
10. Koch, M., Mancini, L.V., Parisi-Presicce, F.: Graph-based specification of access control policies. J. Comput. Syst. Sci. 71(1), 1–33 (2005)
11. König, B., Kozioura, V.: Augur 2 — a new version of a tool for the analysis of graph transformation systems. Electron. Notes Theor. Comput. Sci. 211, 201–210 (2008)
12. Langari, Z., Trefler, R.: Application of graph transformation in verification of dynamic systems. In: Leuschel, M., Wehrheim, H. (eds.) IFM 2009. LNCS, vol. 5423, pp. 261–276. Springer, Heidelberg (2009)

13. Orejas, F., Ehrig, H., Prange, U.: Reasoning with graph constraints. Formal Aspects of Computing 22, 385–422 (2010)
14. Pennemann, K.-H.: Resolution-like theorem proving for high-level conditions. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 289–304. Springer, Heidelberg (2008)
15. Radke, H.: Correctness of graph programs relative to hr+ conditions. In: Ehrig, H., Rensink, A., Rozenberg, G., Schürr, A. (eds.) ICGT 2010. LNCS, vol. 6372, pp. 410–412. Springer, Heidelberg (2010)
16. Strecker, M.: Interactive and automated proofs for graph transformations. Technical report, IRIT/Université de Toulouse (2012), http://www.irit.fr/~Martin.Strecker/Publications/proofs_graph_transformations.html
17. Strecker, M.: Locality in reasoning about graph transformations. In: Schürr, A., Varró, D., Varró, G. (eds.) AGTIVE 2011. LNCS, vol. 7233, pp. 169–181. Springer, Heidelberg (2012)
18. Tran, H.N., Percebois, C.: Towards a rule-level verification framework for property-preserving graph transformations. In: Proceeding of the IEEE ICST Workshop on Verification and Validation of Model Transformations (April 2012)