Locality in Reasoning about Graph Transformations

Martin Strecker

IRIT (Institut de Recherche en Informatique de Toulouse)
Université de Toulouse
http://www.irit.fr/~Martin.Strecker/ *

Abstract. This paper explores how to reason locally about global properties of graph transformations, in particular properties involving transitive closure of the edge relations of the graph. We show under which conditions we can soundly reduce reasoning about all nodes in the graph to reasoning about a finite set of nodes. We then give an effective procedure to turn this reduced problem into a Boolean satisfiability problem.

Keywords: Graph Transformations; Formal Methods; Verification

1 Introduction

Proving the correctness of graph transformations is a notoriously hard problem. As opposed to transformations of tree-like structures, such as term rewriting or functional programming over inductively defined datatypes, there are no well-defined traversal strategies of a graph, and, still worse, multiple matchings of a graph rewriting rule are possible for a single position in a graph. This lack of structure leads to a combinatorial explosion which calls for tool support when proving properties of graph transformations.

Two major approaches have been developed: The *model checking* approach (embodied by [12,3,8,11,19]) considers the evolution of a given start graph under application of transformation rules. Typical questions of interest are whether certain invariants are maintained throughout all evolutions of a graph, or whether certain states are reachable. A downside of this approach is that, in principle, it only allows to talk about the effect of transformations on individual graphs and not about the correctness of rules in general. (However, in particular cases, the rules themselves may generate all interesting instances, such as in the red-black trees of [2].)

The *theorem proving* approach (which we follow in this paper) hoists pre-/postcondition calculi known from imperative programming to programs about graphs. Even before being able to reason about more complex graph transformation programs, one has to be able to prove properties of a single transformation step, *i. e.* the consequences of the application of a single rule.

* Part of this research has been supported by the *Climt* project (ANR-11-BS02-016-02)

Thus, in [6], a relational approach to transformation verification is described and a coding in Event-B presented [17]. In [15], a natural deduction-like proof system for reasoning about graph transformations is given, [9] establishes a correspondence between categorical notions and satisfiability problems, however without providing automated verification procedures.

So far, in all this work, one question has remained unanswered: what is the global impact of applying a rule locally? Under which conditions can reasoning about a graph with an unbounded (possibly even infinite) number of nodes be reduced to reasoning about the finite number of nodes the rule is applied to? The global properties we are particularly interested in are connectedness and reachability, which, in principle, require inductive proofs. We especially concentrate on preservation of transitive closure of the form $r^* \subseteq r'^*$, where r resp. r' are the arc relations of a graph before resp. after the transformation. Finding good answers is essential for automating the verification of graph transformations. We will show when and how we can reduce the containment problem $r^* \subseteq r'^*$ to a problem of the form $r \subseteq r'$. This avoids reasoning about possibly unbounded paths and instead only considers a finite set of nodes and arcs that can be derived directly from the transformation rules. We thus arrive at a problem that can be decided by Boolean satisfiability checking.

We will further highlight the problem and give an informal overview of our approach in Section 2. We then describe our language for specifying graph transformations and the property language in Section 3. In Section 4, we show under which conditions we can separate a graph into an arbitrarily large "exterior" and a finite "interior", and in Section 5, we reduce the verification of a given property on this interior to a Boolean satisfiability problem. In Section 6, we conclude with an outlook on future work.

2 Problem statement

Consider the example in Figure 1, describing a graph transformation rule (in the upper part) that deletes the edge (n_1, n_3) and instead inserts two new edges (n_1, n_2) and (n_2, n_3). When this transformation is embedded into a larger graph (in the lower part; the image of the rule in the graph has a darker shading), one can assert that if it is possible to go from a node x to a node y in the original graph, then this is also possible after transformation. More formally: if r is the edge relation and r^* is its reflexive-transitive closure, then $(x, y) \in r^*$ in the original graph implies $(x, y) \in r^*$ in the transformed graph, for arbitrary nodes x and y.

It might appear that this fact can be established by simply looking at the finite number of nodes of the transformation rule, without taking arbitrary x and y in the graph. Thus, the path $(n_1, n_3) \in r^*$ of the LHS of the rule can be constructed by the composition of (n_1, n_2) and (n_2, n_3) in the RHS, which seems to carry over to embeddings of this rule in larger graphs.

Such a reasoning is fallacious, as illustrated in Figure 2, which is taken from a case study describing the data flow in a communication network (a similar

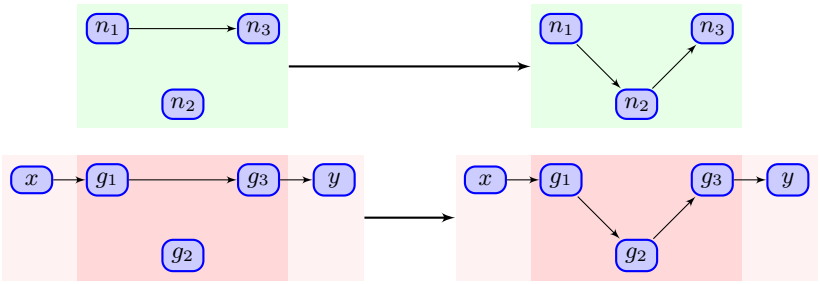

Fig. 1: Simple Rerouting

example appears in [1]). The network consists of composite elements (bold borders in the figure) and simple elements. There are two kinds of relations: the flow relation (simple arrows) and the containment relation between simple and composite elements (dashed arrows). The reader should not be confused by the terminology: also a node representing a composite element is a regular node and not a subgraph.

The transformation rule describes the situation where diagrams with composite elements are flattened, by rerouting data flows to and from composite elements to their containing blocs (of course, the containment relation cannot be part of data paths). One can verify that data paths between non-composite elements are preserved within the rule: the path from n_1 to n_4 in the LHS also exists in the RHS. Unfortunately, data path preservation does not hold any more in a larger context, when considering nodes outside the image of the transformation rule: For example, the data path from g_1 to g_6 is lost by the transformation.

To make the statement ("paths between non-composite blocs") more formal, let us use three kinds of relations: r for edges between simple blocs (straight, non-dashed arrows), and r_{sc} (resp. r_{cs}) for edges from simple to composite (resp. composite to simple) blocs. Let us write $\circ$ for relation composition. The path preservation property now reads: if $(x, y) \in (r \cup (r_{cs} \circ r_{sc}))^*$ in the original graph for arbitrary x, y, then also in the transformed graph.

Why does local reasoning about a rule carry over to the entire graph in the first, but not in the second case? One of the causes appears to be relation composition, which hides an existential quantifier: $(x, y) \in (r_{cs} \circ r_{sc})$ means $\exists z.(x, z) \in r_{cs} \wedge (z, y) \in r_{sc}$, and there is no way to tell whether this z is one of the nodes inside the image of the transformation rule, or whether it lies outside of it. We will therefore omit relation composition from the relational language of Section 5.1, but definitely show that local reasoning is safe for the fragment defined below.

3 Representing and reasoning about transformations

Our way of representing and reasoning about graph transformations is described in more detail in [18], and we therefore only summarize the elements that are

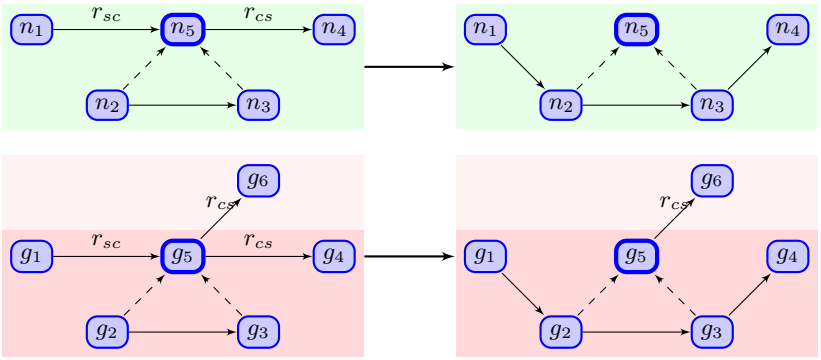

Fig. 2: Complex Rerouting

most important for an understanding of this article. This section might be skipped on a first reading and only be consulted for the terminology used later on.

The approach is entirely logical and consists of coding all relevant notions of graph transformations in a higher-order proof assistant, such as rules, matching, morphisms and rewriting itself. (In our case, the Isabelle [14] system is used, and some of the definitions below are directly extracted from the Isabelle sources.) This gives a fine control over the transformation process, such as the properties of the morphisms mapping rules into graphs. For example, we assume throughout this article that these morphisms are injective, but this property can easily be configured differently, however with a non-negligible impact on efficiency of the reasoning procedures.

A graph is defined to be a record consisting of a set of nodes and a set of edges (pairs of nodes), indexed by an edge type *'et*. Furthermore, nodes can be given a node type:

record (*'nt*, *'et*, *'h*) *graph* =
  nodes :: *obj set*
  edges :: *'et* ⇒ (*obj* ∗ *obj*) *set*
  nodetp :: *obj* ⇒ *'nt*

A graph transformation consists of an applicability condition that defines a matching in a target graph, and an effect that specifies how the matched nodes and edges are transformed: which nodes and edges have to be deleted and which have to be generated, and how the newly generated nodes have to be typed. In order to be able to manipulate the applicability condition syntactically, we do not express it with the aid of the built-in logic of our proof assistant, but rather use a special-purpose syntax that we do not spell out any further here. As an illustration, here is the definition that specifies the transformation of Figure 1. The precondition *rerouting-precond* expresses that there is a type of nodes, *Node*, and a relation *r*, that the nodes *n1*, *n2*, *n3* have type *Node* and there is an edge of type *r* between *n1* and *n3*. The transformation itself specifies which edges are

deleted (*e-del*) and generated (*e-gen*). This specification is indexed by the node relation, hence the lambda-abstraction over *et*. No nodes are deleted or added in this example.

```
datatype nodetp = Node
datatype edgetp = r

definition
  rerouting-precond :: nat ⇒ nat ⇒ nat ⇒ (nodetp, edgetp,'h) path-form where
  rerouting-precond n1 n2 n3 =  (Conjs-form [
     (S-form (Type-set Node) n1),   (S-form (Type-set Node) n2),
     (S-form (Type-set Node) n3),   (P-form (Edge-pth r) n1 n3)  ])

definition rerouting :: (nodetp, edgetp, 'h) graphtrans where
  rerouting =
  (| appcond = rerouting-precond 1 2 3,
    n-del = {},              n-gen = {},
    e-del = λ et. {(1,3)},   e-gen =  λ et. {(1,2), (2,3)},
    n-gentp = empty   |)
```

A morphism is a map from nodes (in the transformation rule) to nodes (in the target graph):

```
types graphmorph = nat ⇒ obj option
```

We now have all ingredients for defining the function *apply-graphtrans-rel* that applies a graph transformation under a given morphism to a graph and produces a transformed graph (see [18] for more details):

```
consts apply-graphtrans-rel ::
  [('nt, 'et, 'h) graphtrans, graphmorph, ('nt, 'et, 'h) graph, ('nt, 'et, 'h) graph]
  ⇒ bool
```

Usually, we are not interested in the behavior of a transformation for one particular morphism, but for any morphism that satisfies the applicability condition. The following relation abstracts away from the morphism and just describes that a graph *gr* is transformed into a graph *gr'* by a graph transformation *gt*:

```
definition  apply-transfo-rel ::
  [('nt, 'et, 'h) graphtrans, ('nt, 'et, 'h) graph, ('nt, 'et, 'h) graph] ⇒ bool where
  apply-transfo-rel gt gr gr' = apply-graphtrans-rel gt (select-morph gt gr) gr gr'
```

The proof obligations we set out to prove typically have the form: *if the transformation transforms gr into gr', then P (gr, gr') holds*, where *P* is a predicate putting into correspondence the node and edge sets of *gr* and *gr'*. For example, the path preservation property of Figure 1 is stated as:

```
(applicable-transfo rerouting gr ∧ apply-transfo-rel rerouting gr gr')
  ⟹  (edges gr r)* ⊆ (edges gr' r)*
```

More specifically, the properties *P* we will examine in the following are simple preservation properties $r \subseteq s$ or path preservation properties $r^* \subseteq s^*$, where r represents the edge relation of *gr* and s the edge relation of *gr'* ("nodes connected

in gr are also connected in gr''"), or inversely ("nodes connected in gr' were already connected in gr", rather expressing a preservation of separation).

The way the transformations are defined, references to gr' can be eliminated. To illustrate this point, let us continue with our example. Unfolding definitions and carrying out some simplifications, our proof obligation reduces to the following (hypotheses are contained in ⟦ ... ⟧, and some inessential ones have been removed):

$$\begin{aligned}&⟦n1 \in nodes\ gr;\ n2 \in nodes\ gr;\ n3 \in nodes\ gr;\ nodetp\ gr\ n1 = Node;\\ &\ nodetp\ gr\ n2 = Node;\ nodetp\ gr\ n3 = Node;\ (n1, n3) \in edges\ gr\ r⟧\\ &\Longrightarrow (edges\ gr\ r)^*\\ &\quad \subseteq (insert\ (n1, n2)\ (insert\ (n2, n3)\ (edges\ gr\ r - \{(n1, n3)\})))^*\end{aligned}$$

4 Graph decompositions

The essential step of our approach consists in splitting up a graph into an interior region (the image of nodes of a rule in a graph under a given graph morphism), and an exterior. Since we are here mostly concerned with the node relations of a graph, we define the interior and exterior of a relation r with respect to a region A as follows:

definition *interior-rel* $A\ r = r \cap (A \times A)$
definition *exterior-rel* $A\ r = r - (A \times A)$

Visually speaking, the interior of a relation wrt. a region A comprises all arcs connecting nodes a_1, a_2 both of which are contained in A, whereas the exterior consists of the arcs connected to at least one node e outside of A.

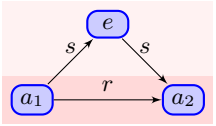

Fig. 3: Interior (dark shade) and exterior (light shade) of a relation

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

lemma *interior-union-exterior*: $interior\text{-}rel\ A\ r \cup exterior\text{-}rel\ A\ r = r$

and from this, we obtain by simple set-theoretic reasoning a first decomposition lemma:

lemma *interior-exterior-decompos-subset-equal*:
$(exterior\text{-}rel\ A\ r \subseteq exterior\text{-}rel\ A\ s \wedge interior\text{-}rel\ A\ r \subseteq interior\text{-}rel\ A\ s)$
$= (r \subseteq s)$

When read from right to left, this lemma reduces reasoning of a relation containment property of the form $r \subseteq s$ to reasoning about the interior and exterior of these relations.

Our principal interest in this paper is reasoning about preservation of (reflexive) transitivity properties, *i. e.* properties of the form $r^* \subseteq s^*$ or $r^+ \subseteq s^+$. Even though applicable in principle, the above decomposition lemma is of no much use, as will become apparent in Section 5. We therefore provide the following specialized form (a similar lemma holds for the transitive closure as well):

lemma *interior-exterior-decompos-rtrancl-subset*:
$(\mathit{exterior\text{-}rel}\ A\ r)^* \subseteq (\mathit{exterior\text{-}rel}\ A\ s)^* \wedge (\mathit{interior\text{-}rel}\ A\ r)^* \subseteq (\mathit{interior\text{-}rel}\ A\ s)^* \Longrightarrow r^* \subseteq s^*$

Its correctness can be seen by using the property $(I^* \cup E^*)^* = (I \cup E)^*$ of transitive closures and lemma *interior-union-exterior* to show that $((\mathit{interior\text{-}rel}\ A\ r)^* \cup (\mathit{exterior\text{-}rel}\ A\ r)^*)^* = r^*$ and then use simple set-theoretic reasoning.

As witnessed by this lemma, proving $r^* \subseteq s^*$ by decomposition into interior and exterior is sound, but contrary to *interior-exterior-decompos-subset-equal*, the above lemma is an implication and not an equality. One may therefore wonder whether this decomposition does not make us lose completeness, by possibly reducing a provable to an unprovable statement.

Indeed, the inverse does not hold in general. For example, $r^* \subseteq s^* \Longrightarrow (\mathit{interior\text{-}rel}\ A\ r)^* \subseteq (\mathit{interior\text{-}rel}\ A\ s)^*$ is not satisfied for $r = \{(a_1, a_2)\}$, $s = \{(a_1, e), (e, a_2)\}$ and $A = \{a_1, a_2\}$. Note that in this case, $\mathit{interior\text{-}rel}\ A\ s = \{\}$, whereas $\mathit{interior\text{-}rel}\ A\ r = \{(a_1,\ a_2)\}$ (also see Figure 3).

However, by choosing the interior large enough, we obtain the inverse of lemma *interior-exterior-decompos-rtrancl-subset* (here, *Field* is the union of the domain and the range of a relation):

lemma *decompos-complete-interior*:
$\mathit{Field}\ s \subseteq A \Longrightarrow r^* \subseteq s^* \Longrightarrow (\mathit{interior\text{-}rel}\ A\ r)^* \subseteq (\mathit{interior\text{-}rel}\ A\ s)^*$
lemma *decompos-complete-exterior*:
$\mathit{Field}\ r \subseteq A \Longrightarrow r^* \subseteq s^* \Longrightarrow (\mathit{exterior\text{-}rel}\ A\ r)^* \subseteq (\mathit{exterior\text{-}rel}\ A\ s)^*$

In practice, this means that the interior has to comprise all known elements of the relations r and s (differently said: all free variables occurring in them).

We have thus recovered completeness of our decomposition for a class of graphs that comprises also the graphs for which this decomposition is in fact trivial (*i. e.* *Field* $s \subseteq A$ and *Field* $r \subseteq A$ hold). It also highlights the problem of performing this decomposition if the fields of the involved relations are not entirely known, as in the case of relation composition.

Let us conclude this section by applying the above procedure to our example. Since the variables *n1*, *n2*, *n3* are free in our goal, we choose $A = \{n1,\ n2,\ n3\}$ and obtain two new subgoals, the first having the conclusion

$(\mathit{exterior\text{-}rel}\ \{n1,\ n2,\ n3\}\ (\mathit{edges}\ gr\ r))^*$
$\subseteq (\mathit{exterior\text{-}rel}\ \{n1,\ n2,\ n3\}$
$\quad (\mathit{insert}\ (n1,\ n2)\ (\mathit{insert}\ (n2,\ n3)\ (\mathit{edges}\ gr\ r - \{(n1,\ n3)\}))))^*$

and the second one being similar, for *interior-rel*.

5 Reduction to Boolean satisfiability

As indicated before, we are interested in proving properties of the form $r \subseteq s$ or $r^* \subseteq s^*$, and we have seen in Section 2 that some relation constructors, such as relation composition, are problematic. In Section 5.1, we therefore give a more precise characterisation of the structure of the relation expressions r and s we will use in the following.

After decomposition of the graph into an exterior and an interior, we are left to simplifying each of these parts separately. In Section 5.2 and Section 5.3 respectively, we will see that for our relation algebra, the resulting problems are indeed decidable.

5.1 Relation expressions

We assume that our relational expressions r are built up inductively according to the following grammar:

$$\begin{aligned} r ::= & \; r_b \\ | & \; \{\} \\ | & \; \mathit{insert}\ (n_1, n_2)\ r \\ | & \; r \cup r \\ | & \; r \cap r \\ | & \; r - r \end{aligned}$$

Here, r_b are basic, non-interpreted relations, and n_1, n_2 are variables representing node names. Our grammar contains set difference $A - B$ and not simply set complement $-B$ because the former behaves better with some of the reductions in Section 5.2 and Section 5.3.

We assume that $FV(r)$ returns all the free variables in r, *i. e.* the set of all the n_1 and n_2 occurring in *insert*-expressions of r.

It should be noted that this grammar enables to capture at least the effects of transformations that arise from application of rules as defined in Section 3.

5.2 Reduction of the exterior

Decomposition of Section 4 has left us with a first subgoal of the form $(\mathit{exterior\text{-}rel}\ A\ r) \subseteq (\mathit{exterior\text{-}rel}\ A\ s)$ or $(\mathit{exterior\text{-}rel}\ A\ r)^* \subseteq (\mathit{exterior\text{-}rel}\ A\ s)^*$. We now exhaustively apply the following rewrite rules (technically, in our framework, they are added to Isabelle's simplification set):

$\mathit{exterior\text{-}rel}\ A\ \{\} = \{\}$
$\mathit{exterior\text{-}rel}\ A\ (\mathit{insert}\ (x,\ y)\ r) = (\{(x,\ y)\} - (A \times A)) \cup\ \mathit{exterior\text{-}rel}\ A\ r$
$\mathit{exterior\text{-}rel}\ A\ (r \cap s) = (\mathit{exterior\text{-}rel}\ A\ r) \cap (\mathit{exterior\text{-}rel}\ A\ s)$

exterior-rel A $(r \cup s) = (\textit{exterior-rel}\ A\ r) \cup (\textit{exterior-rel}\ A\ s)$
exterior-rel A $(r - s) = (\textit{exterior-rel}\ A\ r) - (\textit{exterior-rel}\ A\ s)$
and furthermore the simplification $\{(x, y)\} - (A \times A) = \{\}$ if $x,y \in A$, as well as trivial simplifications of operations with the empty set.

An easy inductive argument shows that, if $FV(r) \subseteq A$, these simplifications reduce any expression *exterior-rel* A r to a Boolean combination, consisting of operators $\cup$, $\cap$, $-$, and only involving expressions *exterior-rel* A r_b, where the r_b are basic relations.

If our original goal had the form $(\textit{exterior-rel}\ A\ r) \subseteq (\textit{exterior-rel}\ A\ s)$, we are now left with a goal $R \subseteq S$, with R and S such Boolean combinations. Since the basic r_b are uninterpreted, this goal can be proved or disproved with propositional reasoning.

If our original goal had the form $(\textit{exterior-rel}\ A\ r)^+ \subseteq (\textit{exterior-rel}\ A\ s)^+$, we are now left with a goal $R^+ \subseteq S^+$, which we can reduce to $R \subseteq S$, using the fact that (reflexive) transitive closure is monotonic, and then proceed as before. This reduction is obviously sound. It is also complete: if $R \subseteq S$ is not provable, any counterexample can be turned into a counterexample in which the basic expressions *exterior-rel* A r_b are interpreted as either empty or non-empty over a one-element domain. Over this domain, a relation and its transitive closure coincide, so that we obtain a counterexample of $R^+ \subseteq S^+$. A slightly more involved argument also holds for reflexive-transitive closure.

5.3 Reduction of the interior

We now show how we can verify the properties in the interior, for goals having the form $(\textit{interior-rel}\ A\ r) \subseteq (\textit{interior-rel}\ A\ s)$ or $(\textit{interior-rel}\ A\ r)^* \subseteq (\textit{interior-rel}\ A\ s)^*$. Even though it appears intuitively plausible that these questions are decidable for a finite interior A, it is not sufficient to use traditional graph algorithms, because our relations r and s are composite.

For simplification, we proceed in several stages:

1. we push inside applications of *interior-rel*, leaving applications of the form *interior-rel* A r_b for basic relations r_b;
2. we explicitly calculate *interior-rel* A r_b, which completely eliminates all occurrences of *interior-rel*;
3. if necessary, we explicitly calculate the (reflexive) transitive closure of the sets thus obtained, which essentially gives a propositional problem;
4. we solve the problem with a propositional solver.

For the first stage, we use the following rewrite rules:

interior-rel A $\{\} = \{\}$
interior-rel A $(\textit{insert}\ (x, y)\ r) = (\{(x, y)\} \cap (A \times A)) \cup (\textit{interior-rel}\ A\ r)$
interior-rel A $(r \cap s) = (\textit{interior-rel}\ A\ r) \cap (\textit{interior-rel}\ A\ s)$
interior-rel A $(r \cup s) = (\textit{interior-rel}\ A\ r) \cup (\textit{interior-rel}\ A\ s)$
interior-rel A $(r - s) = (\textit{interior-rel}\ A\ r) - (\textit{interior-rel}\ A\ s)$

Note that A is a finite, concrete set, so that we can further simplify with:

$x \in A \wedge y \in A \Longrightarrow (\{(x,\ y)\} \cap (A \times A)) = \{(x,\ y)\}$
$x \notin A \vee y \notin A \Longrightarrow (\{(x,\ y)\} \cap (A \times A)) = \{\}$

At the end of this first step, we are now left with a Boolean combination of relations of the style *interior-rel* $A\ r_b$, where r_b is basic. The following equality:

$r \cap (\mathit{insert}\ a\ A) \times (\mathit{insert}\ a\ A) =$
$(r \cap (A \times A)) \cup (r \cap (\{a\} \times (\mathit{insert}\ a\ A))) \cup (r \cap ((\mathit{insert}\ a\ A) \times \{a\}))$

is the basis for a simplification scheme that recurses over A and eliminates all *interior-rel*:

interior-rel $\{\}\ r = \{\}$
interior-rel $(\mathit{insert}\ a\ A)\ r =$
$(\mathit{interior\text{-}rel}\ A\ r) \cup$
$(\mathit{interior\text{-}rel\text{-}elem\text{-}r}\ a\ (\mathit{insert}\ a\ A)\ r) \cup (\mathit{interior\text{-}rel\text{-}elem\text{-}l}\ a\ (\mathit{insert}\ a\ A)\ r)$

Here, we have defined

definition *interior-rel-elem-r* $a\ B\ r = r \cap (\{a\} \times B)$
definition *interior-rel-elem-l* $b\ A\ r = r \cap (A \times \{b\})$

with the following reductions:

interior-rel-elem-r $a\ \{\}\ r = \{\}$
interior-rel-elem-r $a\ (\mathit{insert}\ b\ B)\ r =$
$(\mathit{if}\ (a,\ b) \in r\ \mathit{then}\ \{(a,\ b)\}\ \mathit{else}\ \{\}) \cup \mathit{interior\text{-}rel\text{-}elem\text{-}r}\ a\ B\ r$
interior-rel-elem-l $b\ \{\}\ r = \{\}$
interior-rel-elem-l $b\ (\mathit{insert}\ a\ A)\ r =$
$(\mathit{if}\ (a,\ b) \in r\ \mathit{then}\ \{(a,\ b)\}\ \mathit{else}\ \{\}) \cup \mathit{interior\text{-}rel\text{-}elem\text{-}l}\ b\ A\ r$

Before taking the third step, we have to massage the goal by if-splitting, thus distinguishing, in the above rules, between the cases $(a,\ b) \in r$ and $(a,\ b) \notin r$. This is a source of a considerable combinatorial explosion: for each basic relation r_b occurring in the goal, and for n nodes, we potentially get $2^{(n^2)}$ combinations, the problem being aggravated by the fact that the relations are not directed. We will come back to this point in Section 6.

To complete the transformation, we symbolically compute the closure. We show the recursive equation for transitive closure:

$(v,\ w) \in (\mathit{insert}\ (y,\ x)\ r)^+ =$
$((v,\ w) \in r^+ \vee ((v = y) \vee (v,\ y) \in r^+) \wedge ((x = w) \vee (x,\ w) \in r^+))$

After these transformations, we obtain a Boolean combination of

- membership in an elementary relation: $(x,\ y) \in r$ (or their negation)
- (in)equalities $x = y$ or $x \neq y$ between nodes

This fragment can readily be decided by standard propositional solvers.

6 Conclusions

We have presented a method of automatically proving properties of graph transformations in a restricted relational language with transitive closure, by decomposing the abstract graph in which a rule is to be applied in an interior and an

exterior region on which the proof problems can be verified by essentially propositional means. Seen from a different angle: the present paper establishes a general framework in which the difficultly automizable inductive proofs arising from transitive closure operations have already been carried out on the meta-level. For a particular application, it therefore suffices to use finitary proof methods.

The transformations in Section 5.2 might give rise to problems of set containment for Boolean set operations, which is in principle NP-complete, but seems to be very efficient in practice. The reason is that the simplifications in Section 5.2 only verify that a decomposition has been applied correctly.

However, in their current form, the transformations in Section 5.3 are of exponential complexity. Intuitively, this is because we have to check for the presence of any possible combination of edges in the graph in which the rule has to be applied. It seems difficult to see how to do better, because contrary to tree-like structures, graph transformations allow for a large number of matchings. We are currently working on a proof of the complexity of this decision problem.

We can extend our work in different directions: On the practical side, we would like to develop simplification strategies that perform essential reductions before case splitting introduces a combinatorial explosion.

On the theoretical side, it is interesting to further explore the boundary between decidable and undecidable fragments, hopefully leading to a more expressive relational fragment than the one of Section 5.1. Several logics for reasoning about graphs [5] and for pointer structures in programs have been proposed, and an in-depth comparison still has to be done. The decidable fragment in [10], based on transitive closure logic, only allows for a single edge relation, whereas the logical language in [20] assumes a set of distinguished starting points for talking about reachability, but does not examine reachability between arbitrary points.

The decidable logic described in [13] concentrates on the preservation of data structure variants (such as doubly-linked lists) in pointer manipulating programs, but does not allow to reason about global graph properties.

Separation Logic (SL) [16] is a specialized logic for reasoning in a pre-/ post-condition style about pointer-manipulating programs. It advocates splitting the heap into disjoint areas, thus reducing the complexity of reasoning about pointer manipulations which may have a global impact due to aliasing. Just like Hoare logic, SL as such does not aim at being a decidable logic. The decidable fragment described in [4] is so weak that it is not closed under the heap-manipulating operations of a programming language. It only enables to describe simple properties of lists, but not more general properties of connectivity.

The relation of SL and graph rewriting is discussed in [7], however without reference to particular decision procedures.

Probably the best way to come to terms with the complexity of graph rewriting is to identify particular classes of graphs that are better behaved. This might be tree-like graphs or at least graphs in which the non-determinism of rule application is strongly restricted.

Acknowledgements

I am grateful for my colleagues Ralph Matthes, Christian Percebois and Hanh-Ni Tran for discussions about the problem of locality in graph rewriting and for helpful feedback on preliminary versions of this article.

References

1. Márk Asztalos, László Lengyel, and Tihamer Levendovszky. Towards automated, formal verification of model transformations. In *ICST*, pages 15–24, 2010.
2. Paolo Baldan, Andrea Corradini, Javier Esparza, Tobias Heindel, Barbara König, and Vitali Kozioura. Verifying red-black trees. In *Proc. of COSMICAH '05*, 2005. Proceedings available as report RR-05-04 (Queen Mary, University of London).
3. Paolo Baldan, Andrea Corradini, and Barbara König. A framework for the verification of infinite-state graph transformation systems. *Information and Computation*, 206:869–907, 2008.
4. Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A decidable fragment of separation logic. In *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, pages 97–109, 2004.
5. Ricardo Caferra, Rachid Echahed, and Nicolas Peltier. A term-graph clausal logic: Completeness and incompleteness results. *Journal of Applied Non-classical Logics*, 18(4):373–411, 2008.
6. Simone André da Costa and Leila Ribeiro. Formal verification of graph grammars using mathematical induction. *Electron. Notes Theor. Comput. Sci.*, 240:43–60, July 2009.
7. Mike Dodds. *Graph Transformation and Pointer Structures.* PhD thesis, University of York, 2008.
8. A. H. Ghamarian, M. J. de Mol, A. Rensink, E. Zambon, and M. V. Zimakova. Modelling and analysis using GROOVE. Technical Report TR-CTIT-10-18, Centre for Telematics and Information Technology University of Twente, Enschede, April 2010.
9. Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(02):245–296, 2009.
10. Neil Immerman, Alexander Moshe Rabinovich, Thomas W. Reps, Shmuel Sagiv, and Greta Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *CSL*, pages 160–174, 2004.
11. H. Kastenberg. *Graph-based software specification and verification.* PhD thesis, Univ. of Twente, Enschede, October 2008.
12. Barbara König and Vitali Kozioura. Augur 2—a new version of a tool for the analysis of graph transformation systems. In *Proc. of GT-VMT '06 (Workshop on Graph Transformation and Visual Modeling Techniques)*, volume 211 of *ENTCS*, pages 201–210. Elsevier, 2008.
13. Scott McPeak and George C. Necula. Data structure specifications via local equality axioms. In *Proc. CAV*, pages 476–490, 2005.
14. Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL. A Proof Assistant for Higher-Order Logic.* LNCS 2283. Springer Verlag, 2002.

15. Karl-Heinz Pennemann. Resolution-like theorem proving for high-level conditions. In *Graph Transformations (ICGT'08)*, volume 5214 of *Lecture Notes in Computer Science*, pages 289–304. Springer-Verlag, 2008.
16. John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74, 2002.
17. Leila Ribeiro, Fernando Luís Dotti, Simone André da Costa, and Fabiane Cristine Dillenburg. Towards theorem proving graph grammars using Event-B. *ECEASST*, 30, 2010.
18. Martin Strecker. Modeling and verifying graph transformations in proof assistants. In Ian Mackie and Detlef Plump, editors, *International Workshop on Computing with Terms and Graphs (TERMGRAPH)*, volume 203 of *Electronic Notes in Theoretical Computer Science*, pages 135–148. Elsevier Science, 2008.
19. Dániel Varró and András Balogh. The model transformation language of the VIATRA2 framework. *Science of Computer Programming*, 68(3):214 – 234, 2007. Special Issue on Model Transformation.
20. Greta Yorsh, Alexander Moshe Rabinovich, Mooly Sagiv, Antoine Meyer, and Ahmed Bouajjani. A logic of reachable patterns in linked data-structures. *J. Log. Algebr. Program*, 73(1-2):111–142, 2007.