Electronic Communications of the EASST
Volume 18 (2009)

Proceedings of the Eighth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2009)

Repotting the Geraniums: On Nested Graph Transformation Rules

Arend Rensink and Jan-Hendrik Kuperus

15 pages

Guest Editors: Artur Boronat, Reiko Heckel
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Repotting the Geraniums: On Nested Graph Transformation Rules

Arend Rensink[1] **and Jan-Hendrik Kuperus**[2]

[1]rensink@cs.utwente.nl
Department of Computer Science
Universiteit Twente, Postbus 217, 7500 AE Enschede, The Netherlands

[2]jan-hendrik.kuperus@sogeti.nl
Sogeti Nederland BV, Postbus 76, 4130 EB Vianen, The Netherlands

Abstract: We propose a scheme for rule amalgamation based on nested graph predicates. Essentially, we extend all the graphs in such a predicate with right hand sides. Whenever such an enriched nested predicate matches (i.e., is satisfied by) a given host graph, this results in many individual match morphisms, and thus many "small" rule applications. The total effect is described by the amalgamated rule. This makes for a smooth, uniform and very powerful amalgamation scheme, which we demonstrate on a number of examples. Among the examples is the following, which we believe to be inexpressible in very few other parallel rule formalism proposed in the literature: *repot all flowering geraniums whose pots have cracked.*

Keywords: Geraniums, Graph Transformation, Rule Amalgamation, Quantified Rules, Nested Rules, Parallel Rules

1 Introduction

Standard graph transformation rules are existential. By this we mean that a rule applies wherever there exists a matching of its left hand side into the host graph, and the effect of its application is limited to the homomorphic image of its left hand side under the matching.[1]

The existentiality of rules certainly has advantages, such as their reversibility (at least in a DPO setting). However, there are certain types of transformation where this is clearly a limitation. For instance, if a certain change has to applied *universally*, that is, to *all* sub-graphs with a certain structure, then this can be quite cumbersome to model using existential rules.

This limitation has been recognised especially by tool builders, who after all are in the business of using graph transformations for practical cases; hence, virtually all graph transformation tools have some way to define *rule schemes* or *parallel rules*. (A thorough overview and comparison of related work follows later.) Not all of the solutions have a firm theoretical justification, but the work of Taentzer [28, 26] is ground-breaking in explaining the effect of parallel rule application in a general setting, namely as *rule amalgamation*.

In this paper we describe a way to specify rule amalgamation, based on the concept of *nested graph predicates* of [21, 11]. The basic idea is a very simple one: where a nested graph predicate

[1] An exception to this is the dangling edge deletion by SPO rules; indeed, the fact that this is not existential in the current sense is the reason why such rules cannot be mimicked by single DPO rules.

is essentially a diagram of graphs and graph morphisms, a *nested rule* is a similar diagram of "simple" rules and morphisms. The application of such a rule to a given host graph consists of first matching the graph predicate consisting of the left hand sides of the rule diagram (which will result in a set of match morphisms, each of which goes from a left hand side of one of the simple rules to the host graph), and then using the structure of the rule diagram as interaction scheme in terms of rule amalgamation. The interaction scheme *synchronises* the atomic rule applications according to the match morphisms.

We described a preliminary version of this idea in [22]. This has now been improved and implemented [13], so that we can report on some implementation and performance details. Furthermore, we give some more examples which show the expressiveness of the approach.

The geraniums in the title and abstract refer to the following challenge. We have a number of flower pots, each of which contains a number of geranium plants. These tend to fill all available space with their roots, and so some of the pots have cracked. For each of the cracked pots that contains a geranium that is currently in flower, we want to create a new one, and moreover, to move all flowering plants from the old to the new pot. *Create a single parallel rule that achieves this in a single application, without the use of control expressions.* The complexity of this example stems from the fact that it involves a nested universal quantification, which (as far as we are aware) cannot be expressed in other declarative rule formalisms proposed in the literature, with the possible exception of [9, 14].

The remainder of this paper is structured as follows. In Section 2 we recall the relevant concepts of rule amalgamation; in Section 3 we give a new presentation of nested graph predicates, and we show how these can be used to generate amalgamated rules, which we call nested rules in this paper. In Section 4 we discuss implementation issues, and demonstrate the use of nested rules, including the geraniums as well as some examples encountered in practice. Finally, in Section 5 we discuss related work, draw conclusions and discuss future work.

2 Rule amalgamation

We will first (briefly) recall the concepts of amalgamated graph transformation in the Single Pushout approach from [5], generalising from two rules along the lines of [27], resulting in a setup very similar to [12].

Definition 1 (Graph) A *graph* G is a tuple $\langle V, E, src, tgt \rangle$ consisting of a set of nodes V, a set of edges E, and source and target mappings $src, tgt \colon E \to V$. G is called *labelled* if there is also a function $lab \colon E \to L$ to a global set of labels L, and *simple* if $E \subseteq V \times L \times E$ such that src, lab and tgt are projections to the three components.

The examples in this paper are set in the category of simple labelled graphs, but for the purpose of the definitions one can imagine any pair of graph categories $\mathscr{G}_{\mathsf{tot}}, \mathscr{G}$ such that $\mathscr{G}$ has an initial object 0 and coproducts, and $\mathscr{G}_{\mathsf{tot}}$ is a full subcategory of $\mathscr{G}$ with initial object and coproducts which are preserved by the inclusion functor. We refer to the arrows in $\mathscr{G}$ as partial morphisms and to those in $\mathscr{G}_{\mathsf{tot}}$ as total morphisms.

Recall that a diagram D over a category $\mathscr{C}$ is a mapping from the nodes and edges of a graph

G_D to the objects and arrows of $\mathscr{C}$, such that $D(e)\colon D(src(e)) \to D(tgt(e))$ for all edges e. Diagram D *commutes* if for all parallel paths in G_D, the composition (in $\mathscr{C}$) of the edge images give rise to the same $\mathscr{C}$-arrow. As usual, we will often identify the elements of G_D with their images under D. In this paper we frequently use *tree-shaped diagrams*, in which G_D is a tree rooted in the initial object, i.e., has no cycles, no sharing (no distinct edges with the same target) and exactly one root rt_D (a node without incoming edge) such that $D(rt_D) = 0$. Note that a tree-shaped diagram trivially commutes. A tree-shaped diagram D is said to be en *instance* of another tree-shaped diagram D' if there exists a root-preserving graph morphism $i\colon G_D \to G_{D'}$ (called the *instantiation morphism*) such that $D' = D \circ i$. (So an instance may copy or ignore parts of D.)

Definition 2 (Rule and Sub-rule) A rule is a morphism $p\colon L \to R$ in $\mathscr{G}$. A rule p' is called a *sub-rule* of p if there exists a pair of total morphisms $e_L\colon L' \to L$ and $e_R\colon R' \to R$ such that $p' \circ e_L = e_R \circ p$, i.e., the following diagram commutes:

$$\begin{array}{ccc} L' & \xrightarrow{p'} & R' \\ {\scriptstyle e_L}\downarrow & & \downarrow{\scriptstyle e_R} \\ L & \xrightarrow{p} & R \end{array}$$

The pair $e = e(e_L, e_R)$ is called a *sub-rule embedding*.

Rules give rise to derivations in the usual way of the single-pushout approach.

Definition 3 (Match and Derivation) A *match* of a rule p in a graph G is a total morphism $m\colon L \to G$. Given a rule and a match, a *derivation* is a pushout in $\mathscr{G}$, depicted by the diagram

$$\begin{array}{ccc} L & \xrightarrow{p} & R \\ {\scriptstyle m}\downarrow & \text{PO} & \downarrow{\scriptstyle m'} \\ G & \xrightarrow[d]{} & H \end{array}$$

m' is called the *comatch* and d the *derivation morphism*. Note that m' is in general not total.

We write $G \xrightarrow{p,m} H$ if a derivation as in the above diagram exists. Sub-rules and sub-rule embeddings form a category $\mathscr{R}$ (with the natural definition of identities and arrow composition) with an initial object and coproducts. Below we will sometimes call the objects of $\mathscr{R}$ *simple rules*, to contrast them with the notion of composite rule that we are about to define.

Definition 4 (Composite Rule) A composite rule schema S is a tree-shaped diagram over $\mathscr{R}$.

For instance, Figure 1 shows a composite rule schema that can be used to model the firing of a Petri Net transition. The rule morphisms are left implicit. In general, a composite rule schema corresponds to a *synchronisation rule*, and a composite rule instance (i.e., a tree-shaped diagram P that is an instance of S, in the sense discussed above) to a *component production set* in terms of [27], except that in that paper both kinds of diagrams are required to be bipartite graphs, and the component production set satisfies a certain completeness property.

Every diagram P over $\mathscr{R}$ induces several diagrams over $\mathscr{G}$, among which we will use:

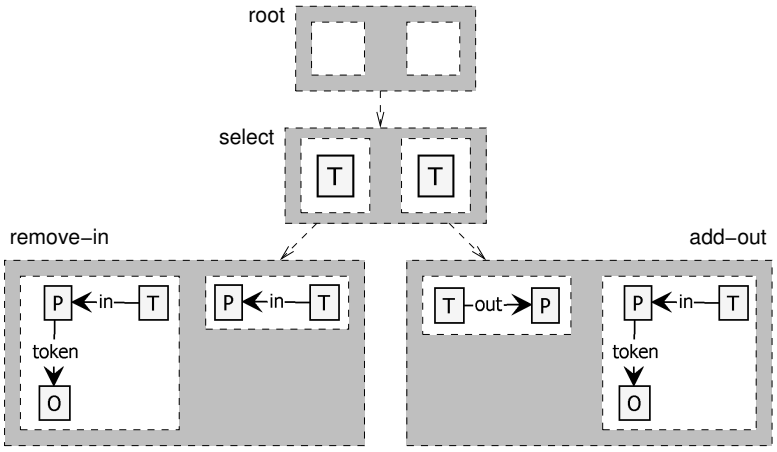

Figure 1: Composite rule schema for firing a Petri Net transition.

- The rule diagram D_P, consisting of the rule morphisms p for all objects p of P and the individual embedding morphisms e_L and e_R for all arrows e of D;
- The "left-hand-side" diagram L_P over $\mathscr{G}_{\mathsf{tot}}$ consisting of all the left hand sides L_p and the corresponding total morphisms e_L.

To define the derivations generated by a composite rule, first we extend the notion of a match.

Definition 5 (Composite Rule Match) Let S be a composite rule schema. A composite rule match of S in G consists of an instance P of S together with a set of matches $m_p\colon L_p \to G$ for all p in P, which, when added to L_P, make the resulting diagram commute.

A match of a composite rule schema S in a graph G is called a (partial) *covering* of G in [27]. Given such a match, as usual one can define the composite derivation (*star-parallel derivation* in [27]) either by taking the coproduct q of the rule diagram D_P and applying that as an ordinary rule (with respect to the unique match of q in G that is guaranteed by the coproduct construction), or by building the coproduct of the diagram consisting of the targets H_p of the individual derivations $G \xrightarrow{p,m_p} H_p$ together with the comatches m'_p. Due to the universal properties of coproducts, these two constructions are guaranteed to yield isomorphic results.

In order to get a useful notion of parallel transformation, the allowed rule schema matches have to be restricted. [27] identifies a number of possible criteria. The main contribution of this paper is to propose yet another criterion, which uses the theory of nested graph predicates introduced by us in [21] and later, independently, in [11].

3 Nested Graph Predicates

We give a new presentation of nested graph predicates, to make the connection with rule amalgamation clearer. A predicate will be a pair consisting of a tree-shaped graph diagram D over $\mathscr{G}_{\mathsf{tot}}$ and a formula generated by the following grammar, $\mathscr{L}$:

$$\phi \;::=\; \mathbf{tt} \mid \neg\phi \mid \phi \vee \phi \mid \exists x.\phi$$

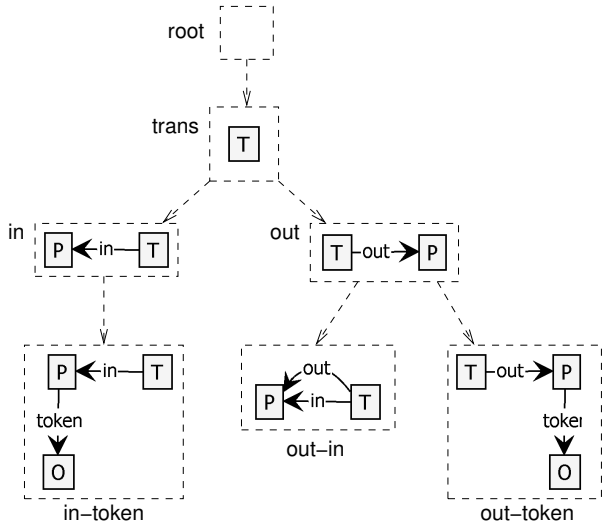

Figure 2: Graph diagram on which transition enabledness can be expressed

Here x denotes one of the non-root nodes of the graph G_D. Apart from the basic logic operators defined above, we also use $\wedge$, $\Rightarrow$, $\forall$ etc., defined in the standard way. Furthermore, we abbreviate $\exists x.\mathbf{tt}$ to x. Every such non-root node x has a unique incoming edge; we will denote this edge in_x.

For instance, some formulae over the diagram in Figure 2 are:

1. $\neg$out-in (which is an abbreviation of $\neg\exists$out-in.$\mathbf{tt}$), expressing that a given Petri Net does not have a loop;
2. $\forall$trans.$\forall$in.in-token, expressing that every transition of a Petri Net can fire;
3. $\exists$trans.$(\forall$in.in-token $\wedge\, \forall$out.$($out-token $\Rightarrow$ out-in$))$, expressing that there is an enabled transition according to the Condition/Event interpretation (in which all output places have to be empty, unless they are also input places).

Formulae are typed over the nodes of G_D. The type of a formula is a graph in D for which we need a matching into the subject graph before we can evaluate the formula; in other words, it represents the "free variables" of the formula. We write $\phi : t$ to denote that t is a type of ϕ. We only deal with formulae that are well-typed according to the following rules:

- $\mathbf{tt} : t$ for all nodes t of G_D;
- $\neg\phi : t$ if $\phi : t$;
- $\phi_1 \vee \phi_2 : t$ if $\phi_1 : t$ and $\phi_2 : t$;
- $\exists x.\phi : t$ if $t = src(in_x)$ and $\phi : x$.

ϕ is called *ground* if $\phi : 0$ (where 0 is the initial object of $\mathscr{G}$). For instance, we have $\neg\exists$out-in.$\mathbf{tt}$: out, whereas the other two example formulae above are ground.

In principle, formulae are evaluated over a given graph G; however, to define this properly we actually have to evaluate them over a given morphism $f : L \to G$, where L is one of the graphs in the diagram D. f in fact represents a matching of L in G that we have built up "so far" while establishing the validity of a larger formula ψ of which ϕ is a sub-formula. The meaning of $\exists x.\phi$

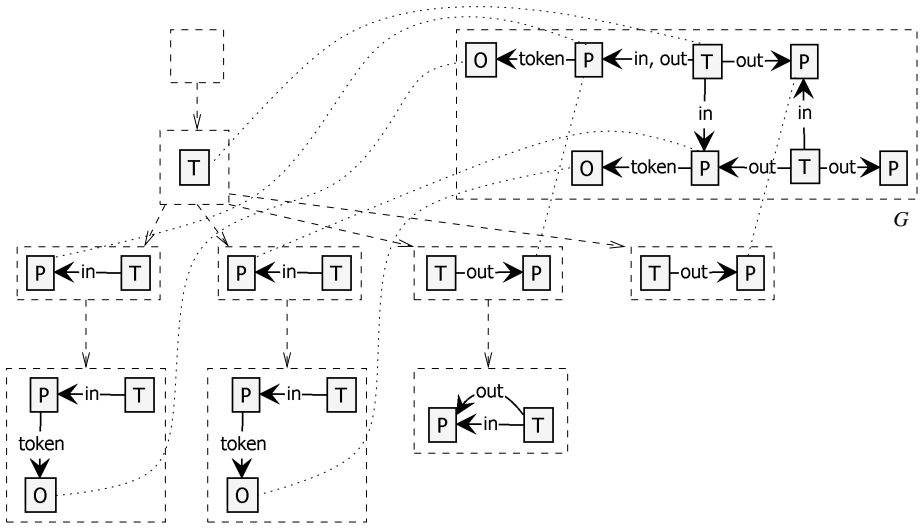

Figure 3: Proof of $\exists$trans.($\forall$in.in-token $\wedge$ $\forall$out.(out-token $\Rightarrow$ out-in)). *The dotted lines indicate some of the relevant node mappings.*

is that the matching f can be decomposed into $g \circ D(in_x)$ (where in_x is the unique edge in D with $tgt(in_x) = x$).

Formally, the semantics of the logic is expressed by a relation $f \models \phi$ where $\phi : t$ and $f\colon D(t) \to G$ is a total morphism in $\mathscr{G}$:

- $f \models \mathbf{tt}$ always holds;
- $f \models \neg\phi$ if $f \not\models \phi$;
- $f \models \phi_q \vee \phi_2$ if $f \models \phi_1$ or $f \models \phi_2$;
- $f \models \exists x : \phi$ if $g \models \phi$ for some g such that $f = g \circ D(in_x)$.

If ϕ is ground, we also write $tgt(f) \models \phi$ instead of $f \models \phi$. For instance, if G is the Petri Net depicted on the right of Figure 3, then the figure shows that there is an enabled Condition/Event transition, as expressed by the example formula 3 above.

A formula ϕ is in positive form if it does not contain negations (but may contain **ff**, $\wedge$ and $\forall$). Every formula is equivalent to a positive form formula, which can be obtained easily by "pushing" negations inward. For instance, formula 3 above is equivalent to $\exists$trans.($\forall$in.$\exists$in-token.**tt** $\wedge$ $\forall$out.($\exists$out-in.**tt** $\vee$ $\forall$out-token.**ff**)).

If ϕ is a ground positive form formula, then a *proof diagram* of $G \models \phi$ is defined to be a commuting diagram P over $\mathscr{G}_{\mathsf{tot}}$, consisting of an instance Q of D with instantiation morphism $i\colon G_Q \to G_D$, augmented with a graph G and for all nodes v of Q a morphism $f_v\colon Q(v) \to G$. Furthermore, for every node v of Q there is a set Ψ_v of sub-formulae of ϕ such that $\phi \in \Psi_{rt_Q}$, and for all $\psi \in \Psi_v$, $\psi : i(v)$ and the following conditions are satisfied:

- $\psi \neq \mathbf{ff}$
- If $\psi = \psi_1 \vee \psi_2$, then either $\psi_1 \in \Psi_v$ or $\psi_2 \in \Psi_v$;
- If $\psi = \psi_1 \wedge \psi_2$, then $\psi_1 \in \Psi_v$ and $\psi_2 \in \Psi_v$;

- If $\psi = \exists x.\psi'$, then v has an outgoing edge e with $i(e) = in_x$ and $\psi' \in \Psi_{tgt(e)}$.
- If $\psi = \forall x.\psi'$, then for all $g\colon D(x) \to G$ such that $f_v = g \circ D(in_x)$, v has an outgoing edge e with $i(e) = in_x$, $f_{tgt(e)} = g$ and $\psi' \in \Psi_{tgt(e)}$.

A proof diagram is called *minimal* if it does not have spurious edges; i.e., the only edges are those necessitated by the last two bullets above. For instance, Figure 3 is a minimal proof diagram, if v and w are the two occurrences of out in the diagram then $\Psi_v = \{\exists \mathsf{out\text{-}in}.\mathbf{tt}\}$ and $\Psi_w = \{\forall \mathsf{out\text{-}token}.\mathbf{ff}\}$.

Predicate-driven amalgamation. The step from nested graph predicates to amalgamated rules is very small: rather than interpreting formulae over diagrams over $\mathscr{G}_{\mathsf{tot}}$, we use tree-shaped diagrams over $\mathscr{R}$, i.e., composite rule schemas. The interpretation of ϕ over S is defined to be its interpretation over the left-hand-side diagram L_S. The following is a key insight:

Proposition 1 *Given a composite rule schema S, a closed formula ϕ interpreted over S, and a graph G, a minimal proof diagram of $G \models \phi$ is a composite match of S in G.*

For instance, we can turn the diagram in Figure 2 into a diagram over $\mathscr{R}$ by replacing the graph in-token by the rule remove-in of Figure 1, replacing out by add-out, and turning all other graphs into identity rules (i.e., based on identity production morphisms). The resulting diagram "refines" Figure 1. The formula $\exists \mathsf{trans}.(\forall \mathsf{in}.\mathsf{in\text{-}token} \wedge \forall \mathsf{out}.(\mathsf{out\text{-}token} \Rightarrow \mathsf{out\text{-}in}))$, which previously just expressed the existence of an enabled transition in a Condition/Event net, now encodes the firing of such a transition under the condition that it is enabled.

The developments in this section culminate in the following definition, which we will use in the remainder of the paper:

Definition 6 (Nested Rule) A nested graph transformation rule is a tree-shaped diagram S over $\mathscr{R}$ with a formula $\phi \in \mathscr{L}$ over S. A match of such a rule is a minimal proof diagram of ϕ over L_S, and a rule derivation is the composite derivation with respect to such a minimal proof diagram.

4 Implementation and examples

The theory of nested rules has been implemented in GROOVE [20], with some restrictions. Nested rules in GROOVE have been used and shown their value in several applications. In this section we discuss some of the implementation choices and show some applications.

4.1 GROOVE implementation

The main functionality of GROOVE is to explore the complete state space of a graph transformation system. Every derivation gives rise to a transition, and independent derivations interleave, giving rise to a size blow-up that is at worst exponential in the number of independent derivations.

A composite rule derivation can combine a large number of simple rule derivations. Apart from the ease of specification, this has the advantage that the number of transitions as well as the number of interleaving points between transitions decreases, in some cases quite dramatically.

For the purposes of practical use, we have made the following choices.

Modified positive form formulae. Rather than the full logic defined above, GROOVE only supports restricted positive form formulae, as defined by the following syntax:

$$\begin{array}{rcl} \phi & ::= & \exists x.(\bigwedge_{k\in K} \neg x_k) \wedge (\bigwedge_{i\in I} \psi_i) \\ \psi & ::= & \forall x.(\bigwedge_{k\in K} \neg x_k) \wedge (\bigvee_{j\in J} \phi_j) \end{array}$$

where $\neg x_k$ abbreviates $\forall x_k.\mathbf{ff}$, and I, J, K are arbitrary index sets. Thus, disjunction is restricted to existentially quantified sub-formulae and conjunction to universally quantified sub-formulae. It can be proved (in fact, it indirectly follows from [21]) that this is no real restriction, in the sense that every formula is equivalent to a "normal form" formula in this restricted syntax, but we will not elaborate on this point here.

Single-graph representation. One of the disadvantages of nested rules as formulated in Definition 6 is that they consist of two parts, a rule diagram and a formula. In GROOVE, we have chosen to include all of these into a single graph representation. For this purpose, we introduce special *quantifier nodes* that stand for the $\forall$- and $\exists$-quantifiers of the formula and are arranged (using special in-labelled edges) in a tree of alternating quantifiers. The root of this tree is an $\exists$-node which is left implicit, so that a simple rule is just a special case of a composite rule.

The "fresh" nodes of the quantified graphs, i.e., those nodes that are not in the codomain of the incoming morphisms, are attached to the corresponding quantifier nodes using special `at`-labelled edges. For fresh edges of the quantified graph, this solution does not work since GROOVE does not support edges on edges; instead, if such a fresh edge does not have fresh end nodes, we include the name of the quantifier as a prefix of the edge label.

As an example, Figure 4 shows the firing rule of Condition/Event nets in this one-graph representation. As usual in the GROOVE notation, non-RHS elements (which are to be deleted) are dashed thin blue (or dark grey), non-LHS elements (which are to be created) are wider solid green (or light grey), and NAC elements are wide, closely dashed red (or dark grey). The dotted nodes and edges form the tree of quantifiers (where the root is omitted); to make the connection with the diagram in Figure 2 explicit, we have named all quantifier nodes. For the existential `out-in-quantifier` this name is in fact necessary as it occurs as a prefix in one of the `in`-edges, to associate this edge with the quantifier.

Non-vacuous universal quantification. If no match of x exists in a given host graph, the formula $\psi = \forall x.\phi$ is true irregardless of ϕ. In this case, ψ is said to be *vacuously true*. Consequently, a universally quantified nested (sub-)rule may be vacuously applicable, in which case the rule has no effect. Sometimes this may be just what one wants, as in the firing rule of Figure 4: for a transition with no input places, the sub-rule `in` is always enabled and has no effect. However, quite often vacuous derivations are not intended. Though non-vacuity can always be enforced through an application condition, we have included a special quantifier node, denoted $\forall^{>0}$, which guarantees that the sub-rule is matched at least once. Thus, $\forall^{>0}x.\phi$ is equivalent to $\exists x.\phi \wedge \forall x.\phi$.[2]

[2] Thus, the difference between $\forall$ and $\forall^{>0}$ is very similar to that between optional and obligatory set nodes in PROGRES.

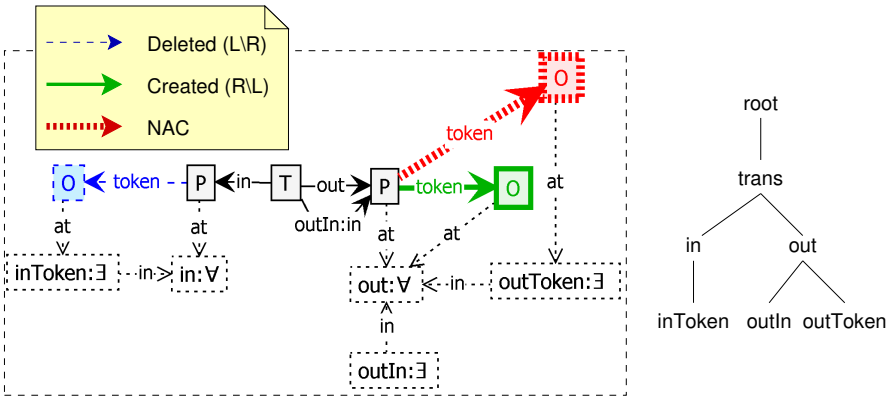

Figure 4: Nested C/E firing rule in GROOVE *syntax, with the explicit tree structure shown to the right*

4.2 Examples

We now show some other applications of nested rules.

Geraniums. The title challenge of this paper is to create a new pot for every cracked flower pot with at least one flowering geranium, and to transfer all flowering geraniums in the cracked pot to the new one. This is an example of a rule that needs two nested universal quantifiers: an outer quantifier for the pots, and an inner quantifier for the plants in the pots. This puts the rule beyond what can be formulated in other approaches to parallel graph transformation rules, such as the cloning rules of [18] or the set nodes and star rules in PROGRES [24], except in the extension recently proposed in [9] — see Section 5 for a more extensive discussion.

In GROOVE, a first attempt is given on the left side of Figure 5. An example derivation is shown in Figure 6. However, this rule is incorrect as it also creates new pots for cracked pots that do not contain any flowering geraniums. To rule this out, we need the non-vacuous universal quantifier discussed above. However, we cannot simply replace the plants-quantifier by $\forall^{>0}$, since then the rule requires that *all* cracked pots have at least one flowering geranium, hence it would become inapplicable for a graph like the one in Figure 6. To resolve this, we have to add

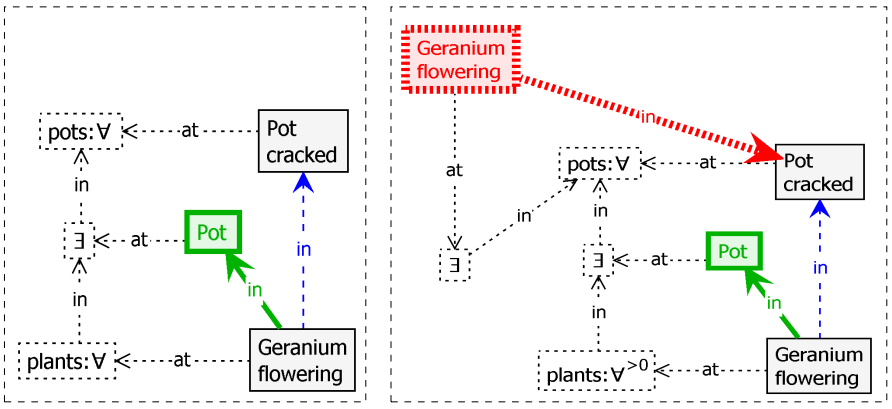

Figure 5: Incorrect and corrected versions of the geranium rule.

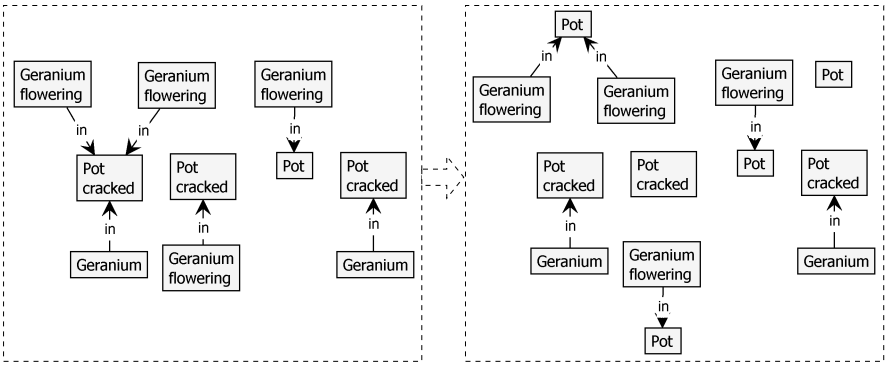

Figure 6: Example derivation of the left hand rule of Figure 5

a disjunct to the pots-quantifier, resulting in the rule on the right hand side of Figure 5.

Sierpinski Triangles Another example that is very suited to specification in nested rules is the Sierpinski case described in [29]. This involves a challenge to give a graph grammar that generates all Sierpinski triangles (a certain fractal shape) up to an arbitrary depth. One step of the generation process involves replacing all up-pointing sub-triangles by a more involved graph (which contains three new up-pointing triangles). A nested GROOVE rule that specifies this given in Figure 7.

In [29], we have described a sequential GROOVE solution to the Sierpinski case, and we have remarked that the above parallel rule has (only) slightly better performance. This may be surprising in the light of the fact that the sequential solution generates many more intermediate states. However, in this particular case no real state space exploration is needed: instead, a "linear" exploration strategy is used that selects a single rule application and never backtracks. In this type of exploration, generating the intermediate states causes only little overhead.

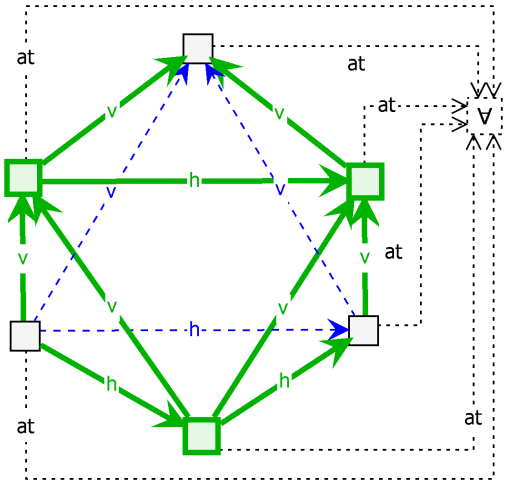

Figure 7: Nested rule specifying one Sierpinski triangle generation step.

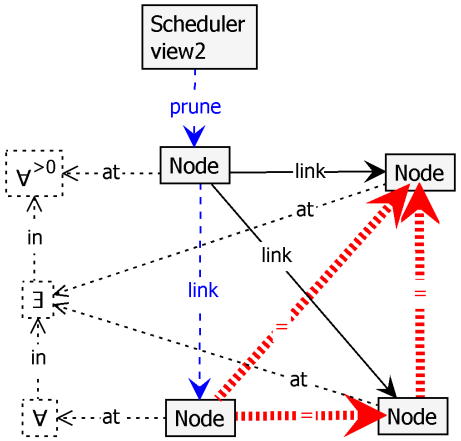

Figure 8: One of the rules of the ad-hoc network connectivity protocol in [3]. The =-labelled NAC-edges are injectivity constraints.

Network gossipping protocol. In [3] we describe a GROOVE model of an ad-hoc network connectivity protocol. The paper shows that this model gives rise to a large symmetry reduction, so that larger network instances can be modelled than with other specification methods (although the size of the state space is still exponential in the size of the network). Nested rules have been used here in several places, to reduce the number of derivation steps and especially the number of interleavings of steps. In contrast to the previous example, in this case it is important to explore the state space in full, and indeed without the use of nested rules the advantage with respect to other methods to some degree disappears. An example rule where nesting has been exploited is shown in Figure 8: this specifies that all but two outgoing link-edges have to be removed from every network node.

5 Evaluation

We have shown how to integrate the concepts of nested graph predicates and rule amalgamation. It turns out that these concepts mesh together quite well, and give rise to a usable specification formalism for parallel rules. This formalism has been implemented in GROOVE; we have given several practical examples where this type of rule has been very useful.

On the downside, it turns out that nested rules can be complicated to write. This is mainly due to the chosen single-graph representation: especially when the rules become larger, the fact that all nesting levels are combined in a single graph makes the resulting figure hard to read. An alternative is to use a hierarchical graph syntax, where the quantifier nodes are containers for the graph elements associated with them.

Related work. We briefly review alternative approaches to parallel rule specification.

First of all, node replacement systems [7] have a natural notion of parallelism due to the fact that, when a node is replaced, all incident edges, no matter how many, are modified as well. For the *star grammars* [4] this is generalised so that not only incident edges but their opposite nodes can be duplicated as often as necessary. This roughly corresponds a single universal quantifica-

tion in terms of our nested rules; in fact, every adaptive star rule can easily be formulated as a nested rule with a single universal quantifier.

PROGRES [24] and also FuJaBA [19] feature so-called *set nodes*, which are essentially single universally quantified nodes. Furthermore, PROGRES has *star rules*, which are essentially rules that are entirely universally quantified. An interesting extension to set nodes can be found in [9], which allows to specify *set regions* rather than just set nodes. Since these set regions can be nested, this comes close to our notion of nested quantifiers, and we conjecture that this formalism can in fact specify the geranium rule. Unfortunately, the paper does not provide enough information to be sure.

Some approaches are based on rule schemas with sub-graphs that can be cloned or copied before applying the rule: for instance, [18, 1, 17]. The latter two have the interesting option of specifying connections *between* the clones, which may for instance be ordered in a linear list. This is outside the capabilities of our nested rules. On the other hand, each of these approaches deals with a single level of (universal) quantification only, and so we believe that they cannot solve the geranium challenge.

As we have made clear, our nested rules are built on the principle of rule amalgamation. Other papers that have shown the power of amalgamation for specifying parallel rules (in particular, the Petri net firing rule) are [15, 8].

Another approach that needs mentioning in this context is that of *synchonised hyperedge replacement*; see, e.g., [14]. A central concept in this formalism is to combine "local" rules into larger ones, using *synchonisation algebras* to determine how rules are to be combined. It is claimed in [30] that this is powerful enough to repot the geraniums.

Finally, another method altogether for repotting the geraniums is by using control expressions rather than a single parallel rule or rule schema. There have been many proposals for powerful control languages; we would like to mention PROGRES, FuJaBA's storyboards, but also the recent notions of *recursive rules* [10, 32], which in fact have no extraneous control conditions but rather integrate them with the rules themselves. We would also categorise the use of the (very powerful) pattern definitions in model transformation tools such as TEFKAT [16] and VIATRA2 [31] as control expressions, though admitedly the dividing line grows thin in these cases.

Future work. We see nested rules as a first step towards the ability to specify an arbitrary transformation as a single transaction with atomic execution. The planned next step is to enhance the GROOVE control language with an atomicity statement that turns an arbitrary control statement into such a transaction.

Another potentially useful extension is the introduction of *counting quantifiers*, being existential quantifiers that assert the existence of a given, fixed number of distinct instances of a sub-graph (other than 1, which is the default meaning of existential quantification). For instance, the rule in Fig. 8 could be simplified using such a feature.

Bibliography

[1] D. Balasubramanian, A. Narayanan, S. Neema, F. Shi, R. Thibodeaux, and G. Karsai. A subgraph operator for graph transformation languages. In K. Ehrig and H. Giese [6].

[2] A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, eds. *Third International Conference on Graph Transformations (ICGT)*, vol. 4178 of *LNCS*. Springer, 2006.

[3] P. Crouzen, J. van de Pol, and A. Rensink. Applying formal methods to gossiping networks with mCRL and Groove. *SIGMETRICS Perform. Eval. Rev.*, 36(3):7–16, 2008.

[4] F. Drewes, B. Hoffmann, D. Janssens, M. Minas, and N. V. Eetvelde. Adaptive star grammars. In A. Corradini et al. [2], pp. 77–91.

[5] H. Ehrig and M. Löwe. Parallel and distributed derivations in the single-pushout approach. *TCS*, 109(1&2):123–143, 1993.

[6] K. Ehrig and H. Giese, eds. *Graph Transformation and Visual Modeling Techniques (GT-VMT)*, vol. 6 of *Electronic Communications of the EASST*. EASST, 2007.

[7] J. Engelfriet and G. Rozenberg. Node replacement graph grammars. In G. Rozenberg [23], pp. 1–94.

[8] C. Ermel, G. Taentzer, and R. Bardohl. Simulating algebraic high-level nets by parallel attributed graph transformation. In Kreowski, Montanari, Orejas, Rozenberg, and Taentzer, eds., *Formal Methods in Software and Systems Modeling*, vol. 3393 of *LNCS*, pp. 64–83. Springer, 2005.

[9] C. Fuss and V. E. Tuttlies. Simulating set-valued transformations with algorithmic graph transformation languages. In A. Schürr et al. [25], pp. 442–455.

[10] E. Guerra and J. de Lara. Adding recursion to graph transformation. In K. Ehrig and H. Giese [6].

[11] A. Habel and K.-H. Pennemann. Nested constraints and application conditions for high-level structures. In Kreowski, Montanari, Orejas, Rozenberg, and Taentzer, eds., *Formal Methods in Software and Systems Modeling*, vol. 3393 of *LNCS*, pp. 293–308. Springer, 2005.

[12] R. Heckel, J. Müller, G. Taentzer, and A. Wagner. Attributed graph transformations with controlled application of rules. In Valiente and Rossello Llompart, eds., *Colloquium on Graph Transformation and its Application in Computer Science*, Technical Report B–19. Universitat de les Illes Balears, 1995.

[13] J.-H. Kuperus. Nested quantification in graph transformation rules. Master's thesis, Department of Computer Science, University of Twente, 2006.

[14] I. Lanese and E. Tuosto. Synchronized hyperedge replacement for heterogeneous systems. In Jacquet and Picco, eds., *Coordination Models and Languages (COORDINATION)*, vol. 3454 of *LNCS*, pp. 220–235. Springer, 2005.

[15] J. de Lara, C. Ermel, G. Taentzer, and K. Ehrig. Parallel graph transformation for model simulation applied to timed transition Petri Nets. In Heckel, ed., *Graph Transformations and Visual Modelling Techniques (GT-VMT)*, vol. 109 of *ENTCS*, pp. 17–29, 2004.

[16] M. Lawley and J. Steel. Practical declarative model transformation with tefkat. In Bruel, ed., *MoDELS Satellite Events*, vol. 3844 of *LNCS*, pp. 139–150. Springer, 2006.

[17] J. Lindqvist, T. Lundkvist, and I. Porres. A query language with the star operator. In K. Ehrig and H. Giese [6].

[18] M. Minas and B. Hoffmann. An example of cloning graph transformation rules for programming. In R. Bruni and D. Varró, eds., *Graph Transformation and Visual Modeling Techniques (GT-VMT)*, vol. 211 of *ENTCS*, pp. 241–250, 2006.

[19] J. Niere and A. Zündorf. Using FUJABA for the development of production control systems. In Nagl, Schürr, and Münch, eds., *Applications of Graph Transformations with Industrial Relevance, (AGTIVE)*, vol. 1779 of *LNCS*, pp. 181–191. Springer, 2000.

[20] A. Rensink. The GROOVE simulator: A tool for state space generation. In Pfaltz, Nagl, and Böhlen, eds., *Applications of Graph Transformations with Industrial Relevance, (AGTIVE)*, vol. 3062 of *LNCS*, pp. 479–485. Springer, 2004.

[21] A. Rensink. Representing first-order logic using graphs. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, eds., *Second International Conference on Graph Transformations (ICGT)*, vol. 3256 of *LNCS*, pp. 319–335. Springer, 2004.

[22] A. Rensink. Nested quantification in graph transformation rules. In A. Corradini et al. [2], pp. 1–13.

[23] G. Rozenberg, ed. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[24] A. Schürr. Programmed graph replacement systems. In G. Rozenberg [23], pp. 479–546.

[25] A. Schürr, M. Nagl, and A. Zündorf, eds. *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, vol. 5088 of *LNCS*. Springer, 2008.

[26] G. Taentzer. *Parallel and Distributed Graph Transformation: Formal Description and Application to Communication-Based Systems*. PhD thesis, TU Berlin, 1996.

[27] G. Taentzer. Parallel high-level replacement systems. *TCS*, 186(1-2):43–81, 1997.

[28] G. Taentzer and M. Beyer. Amalgamated graph transformations and their use for specifying AGG — an algebraic graph grammar system. In Schneider and Ehrig, eds., *Graph Transformations in Computer Science*, vol. 776 of *LNCS*, pp. 380–394. Springer, 1994.

[29] G. Taentzer, E. Biermann, D. Bisztray, B. Bohnet, I. Boneva, A. Boronat, L. Geiger, R. Geiß, Á. Horvath, O. Kniemeyer, T. Mens, B. Ness, D. Plump, and T. Vajk. Generation of Sierpinski triangles: A case study for graph transformation tools. In A. Schürr et al. [25], pp. 514–539.

[30] E. Tuosto. Private communication, 2009.

[31] D. Varró and A. Balogh. The model transformation language of the VIATRA2 framework. *Sci. Comput. Program.*, 68(3):214–234, 2007.

[32] G. Varró, Á. Horváth, and D. Varró. Recursive graph pattern matching with magic sets and global search plans. In A. Schürr et al. [25], pp. 456–470.