Pattern-Based Graph Abstraction[⋆]

Arend Rensink and Eduardo Zambon

Formal Methods and Tools Group,
Computer Science Department,
University of Twente
P.O. Box 217, 7500 AE, Enschede, The Netherlands
{rensink,zambon}@cs.utwente.nl

Abstract. We present a new abstraction technique for the exploration of graph transformation systems with infinite state spaces. This technique is based on *patterns*, simple graphs describing structures of interest that should be preserved by the abstraction. Patterns are collected into *pattern graphs*, layered graphs that capture the hierarchical composition of smaller patterns into larger ones. Pattern graphs are then abstracted to a finite universe of *pattern shapes* by collapsing equivalent patterns. This paper shows how the application of production rules can be lifted to pattern shapes, resulting in an over-approximation of the original system behaviour and thus enabling verification on the abstract level.

1 Introduction

Graph transformation (GT) is a framework that, on one hand, is intuitive and flexible enough to serve as a basic representation of many kinds of structures, and, on the other hand, is precise and powerful enough to formally describe system behaviour. Many techniques and tools have been proposed to analyse the behaviour of GT systems. In particular, systems with infinite state spaces pose a challenge since they require some form of abstraction. A key aspect when designing such abstractions is the trade-off between preserving the expressive power of GT and managing the complexity of the abstraction mechanism. This trade-off has been considered at various points of its scale on the different abstractions given in the literature [1,3,17].

In this paper we present a new abstraction based on *graph patterns* (simple edge-labelled graphs), to be used in the analysis of infinite-state GT systems. The novelty of the approach lies in the flexibility for tuning the abstraction according to the substructures of interest, represented via a *type graph*. At the concrete level we work with *pattern graphs*, a layered structure that describes the composition of patterns. The abstraction of pattern graphs gives rise to *pattern shapes*, which are bounded structures forming a finite universe.

In this work we define how pattern graphs and pattern shapes are constructed from simple graphs, and we show how the application of GT rules can be lifted

⋆ The work reported herein is being carried out as part of the GRAIL project, funded by NWO (Grant 612.000.632).

to these new structures. Our major result is a proof that the abstract state space obtained by transforming pattern shapes is an over-approximation of the original system behaviour, thus enabling verification on the abstract level. This text is an abridged version of [16] where all technical details, including proofs, are given.

2 Simple Graphs

In its basic form, a graph is composed of nodes and directed binary edges.

Definition 1 (graph). *A* graph *is a tuple* $G = \langle N_G, E_G, \mathsf{src}_G, \mathsf{tgt}_G \rangle$ *where*

- N_G *is a finite set of nodes;*
- E_G *is a finite set of edges, disjoint from* N_G*; and*
- $\mathsf{src}_G : E_G \to N_G$ *and* $\mathsf{tgt}_G : E_G \to N_G$ *are mappings associating each edge to its source and target nodes, respectively.* ◀

For a node $v \in N_G$, we consider the set of edges *outgoing* from and *incoming* to v, defined as $v{\rhd}_G = \{e \in E_G \mid \mathsf{src}_G(e) = v\}$ and $v{\lhd}_G = \{e \in E_G \mid \mathsf{tgt}_G(e) = v\}$, respectively. A *path* in G is a non-empty sequence of edges $\pi = e_1 \cdots e_k$ such that $\mathsf{tgt}_G(e_i) = \mathsf{src}_G(e_{i+1})$ for $1 \le i < k$. For convenience, we write $\mathsf{src}(\pi) = \mathsf{src}_G(e_1)$ and $\mathsf{tgt}(\pi) = \mathsf{tgt}_G(e_k)$. Paths π_1, π_2 are *parallel* if $\mathsf{src}(\pi_1) = \mathsf{src}(\pi_2)$ and $\mathsf{tgt}(\pi_1) = \mathsf{tgt}(\pi_2)$. Furthermore, v is a *predecessor* of w in G, denoted $v \le_G w$, if either $v = w$ or there is a path π with $\mathsf{src}(\pi) = v$ and $\mathsf{tgt}(\pi) = w$.

Definition 2 (graph morphism). *A* graph morphism *between graphs* G, H *is a function* $m : (N_G \cup E_G) \to (N_H \cup E_H)$*, such that* $m(N_G) \subseteq N_H$*,* $m(E_G) \subseteq E_H$*,* $\mathsf{src}_H \circ m = m \circ \mathsf{src}_G$*, and* $\mathsf{tgt}_H \circ m = m \circ \mathsf{tgt}_G$*.* ◀

If function m is injective (surjective, bijective) then the morphism is called injective (surjective, bijective). A bijective morphism is also called an *isomorphism* and we write $G \simeq H$ to denote that there is an isomorphism between G and H. We use $m : G \to H$ as a short-hand notation for $m : (N_G \cup E_G) \to (N_H \cup E_H)$. If $G \subseteq H$, we use $\mathsf{emb}(G, H)$ to denote the *embedding* of G into H.

Let Lab be a finite set of labels, partitioned into disjoint unary and binary label sets, denoted $\mathsf{Lab}^{\mathsf{U}}$ and $\mathsf{Lab}^{\mathsf{B}}$, respectively.

Definition 3 (simple graph). *A* simple graph *G is a graph extended with an edge labelling function* $\mathsf{lab}_G : E_G \to \mathsf{Lab}$*, where*

- *for all* $e \in E_G$*, if* $\mathsf{lab}_G(e) \in \mathsf{Lab}^{\mathsf{U}}$ *then* $\mathsf{src}_G(e) = \mathsf{tgt}_G(e)$*; and*
- *for any* $e_1, e_2 \in E_G$*, if* $\mathsf{src}_G(e_1) = \mathsf{src}_G(e_2)$*,* $\mathsf{tgt}_G(e_1) = \mathsf{tgt}_G(e_2)$*, and* $\mathsf{lab}_G(e_1) = \mathsf{lab}_G(e_2)$*, then* $e_1 = e_2$*.* ◀

The second condition in the definition above prohibits parallel edges with the same label, justifying the choice of the term simple graph. The first condition limits the occurrence of unary labels to self-edges, which are used to encode node labels. We write $\langle v, l, w \rangle$ to represent an edge e with $\mathsf{src}_G(e) = v$, $\mathsf{lab}_G(e) = l$, and $\mathsf{tgt}_G(e) = w$. The universe of simple graphs is denoted by SGraph.

Figure 1(a) shows an example of a simple graph representing a single-linked list composed of five cells and a sentinel node to mark the head and tail elements

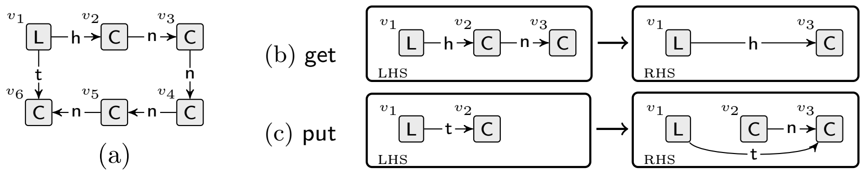

Fig. 1. (a) Simple graph representing a linked list. (b,c) Two transformation rules.

of the list. Unary labels are shown inside their associated node and node identities are displayed at the top left corner of each node.

Definition 4 (simple graph morphism). *A* simple graph morphism *between simple graphs $G, H \in \mathsf{SGraph}$ is a graph morphism $m : G \to H$ that preserves edge labels,* i.e.*, $\mathsf{lab}_H \circ m = \mathsf{lab}_G$.* ◀

We write SMorph to represent the universe of simple graph morphisms. Simple graphs are transformed by simple graph rules.

Definition 5 (simple graph transformation rule). *A* simple graph transformation rule *$r = \langle L, R\rangle$ consists of two simple graphs $L, R \in \mathsf{SGraph}$, called left-hand side (*LHS*) and right-hand side (*RHS*), respectively.* ◀

The relation between L and R is established by their common elements, via an implicit identity morphism on L. We distinguish the following sets:

– $N^{\mathsf{del}} = N_L \setminus N_R$ and $E^{\mathsf{del}} = E_L \setminus E_R$ are the sets of elements deleted; and
– $N^{\mathsf{new}} = N_R \setminus N_L$ and $E^{\mathsf{new}} = E_R \setminus E_L$ are the elements created by the rule.

We write $U = L \cup R$ and $U^{\mathsf{new}} = N^{\mathsf{new}} \cup E^{\mathsf{new}}$ as short-hand notation, and we use SRule to denote the universe of simple graph transformation rules. Figure 1(b,c) shows rules removing the head element of a list (get) and inserting a new element at the tail of the list (put).

Definition 6 (simple graph transformation). *Let G be a simple graph and $r = \langle L, R\rangle$ a simple graph transformation rule such that G and U^{new} are disjoint. An* application *of r into G involves finding a* match *m of r into G, which is an* injective *simple graph morphism $m : L \to G$. Extend m to U by taking $m \cup \mathsf{id}_{U^{\mathsf{new}}}$. Given such m, rule r* transforms *G into a new simple graph H, where*

– $N_H = (N_G \setminus m(N^{\mathsf{del}})) \cup N^{\mathsf{new}}$;
– $E_H = (\{e \in E_G \mid \mathsf{src}_G(e), \mathsf{tgt}_G(e) \notin m(N^{\mathsf{del}})\} \setminus m(E^{\mathsf{del}})) \cup E^{\mathsf{new}}$;
– $\mathsf{src}_H = (\mathsf{src}_G \cup (m \circ \mathsf{src}_U))|_{E_H}$, $\mathsf{tgt}_H = (\mathsf{tgt}_G \cup (m \circ \mathsf{tgt}_U))|_{E_H}$; *and*
– $\mathsf{lab}_H = (\mathsf{lab}_G \cup \mathsf{lab}_U)|_{E_H}$. ◀

In the definition above, dangling edges are deleted, following the SPO approach. Since we do not distinguish between isomorphic graphs, the assumption that U^{new} and G are disjoint can be satisfied without loss of generality by taking an isomorphic copy of U where the elements of U^{new} are fresh with respect to G. We write $G \xrightarrow{r} H$ to denote that the application of r to G (under some m) gives rise to the transformed graph H.

Definition 7 (simple graph grammar). *A* simple graph grammar *is a tuple $\mathcal{G} = \langle \mathcal{R}, G_0 \rangle$, with $\mathcal{R}$ a set of simple graph rules and G_0 an initial simple graph.* ◀

Our standard model of behaviour is a simple graph transition system (SGTS).

Definition 8 (SGTS). *A* simple graph transition system *is a tuple $SGTS = \langle \mathcal{S}, \rightarrow, \iota \rangle$ where $\mathcal{S} \subseteq \mathsf{SGraph}$ is a set of states, $\rightarrow \subseteq \mathcal{S} \times \mathsf{SRule} \times \mathcal{S}$ a set of rule applications, and $\iota \in \mathcal{S}$ is the initial state. Grammar $\mathcal{G}$* generates *a $SGTS_{\mathcal{G}}$ if $\iota = G_0$ and $\mathcal{S}$ is the minimal set of graphs such that $G \in \mathcal{S}$ and $G \xrightarrow{r} H$ for $r \in \mathcal{R}$ implies that there exists $H' \in \mathcal{S}$ where $H \simeq H'$ and $G \xrightarrow{r} H'$ is a transition.* ◀

3 Pattern Graphs

Pattern graphs are the cornerstone graph representation in this work. We first give a general definition, but in this paper we restrict ourselves to pattern graphs that are *well-formed*, a condition that will be presented shortly.

Definition 9 (pattern graph). *A* pattern graph *P is a graph extended with a labelling function $\mathsf{lab}_P : N_P \cup E_P \rightarrow \mathsf{SGraph} \cup \mathsf{SMorph}$ that maps nodes to simple graphs ($\mathsf{lab}_P(N_P) \subseteq \mathsf{SGraph}$), and edges to simple graph morphisms ($\mathsf{lab}_P(E_P) \subseteq \mathsf{SMorph}$), such that $\mathsf{lab}_P(d) : \mathsf{lab}_P(\mathsf{src}_P(d)) \rightarrow \mathsf{lab}_P(\mathsf{tgt}_P(d))$ is an* injective, non-surjective *simple graph morphism, for all $d \in E_P$.* ◀

In categorical terms, a pattern graph corresponds to a diagram in the category SGraph. Elements of N_P and E_P are called *pattern nodes* and *pattern edges*, respectively. For $p \in N_P$, $\mathsf{lab}_P(p)$ is the *pattern* of p. As with simple graphs, we may write $\langle p, f, q \rangle$ as a short-hand for a pattern edge d with $\mathsf{src}_P(d) = p$, $\mathsf{lab}_P(d) = f$, and $\mathsf{tgt}_P(d) = q$. Note that the restriction to non-surjective simple graph morphisms means that the pattern of $\mathsf{tgt}_P(d)$ is always strictly larger than that of $\mathsf{src}_P(d)$, which in turn implies that a pattern graph is always a dag. We layer the nodes of P according to the number of simple edges in their patterns (for $i \geq 0$): $N_P^i = \{p \in N_P \mid |E_G| = i, G = \mathsf{lab}_P(p)\}$ and $N_P^{i+} = \bigcup_{j \geq i} N_P^j$.

Figure 2 shows an example of a pattern graph. Pattern nodes are drawn with dashed lines and the associated patterns are shown inside the node. Pattern edges are depicted as arrows labelled with their corresponding simple graph morphisms, except that embeddings are omitted to avoid clutter. Layers are indicated on the right. Note that there is no distinction between simple edges with unary or binary labels for the purpose of layer assignment. From here on we simplify the figures by showing only the patterns with labelled simple nodes.

Let $d = \langle p, f, q \rangle \in E_P$ be a pattern edge and let $G = \mathsf{lab}_P(p)$. The *image* of d is defined as $\mathsf{img}_P(d) = H$ where $N_H = f(N_G)$, $E_H = f(E_G)$, $\mathsf{src}_H = f \circ \mathsf{src}_G$, $\mathsf{tgt}_H = f \circ \mathsf{tgt}_G$, and $\mathsf{lab}_H \circ f = \mathsf{lab}_G$. It is easy to see that $H \subset \mathsf{lab}_P(q)$. We say that every pattern edge d *covers* the sub-graph $\mathsf{img}_P(d)$. Furthermore, we call a set of pattern edges $E' \subseteq E_P$ *jointly surjective* if $\mathsf{tgt}_P(d_1) = \mathsf{tgt}_P(d_2) = p$ for all $d_1, d_2 \in E'$, and $\bigcup_{d \in E'} \mathsf{img}_P(d) = \mathsf{lab}_P(p)$. As a equivalent term we say that the pattern edges of E' *together cover* $\mathsf{lab}_P(p)$.

A pattern graph P is called *well-formed* if (with $G = \mathsf{lab}_P(p)$)

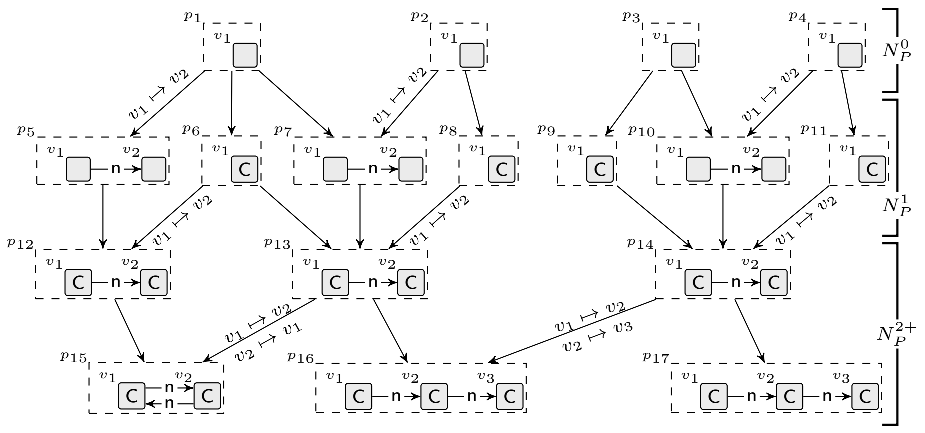

Fig. 2. Example of a pattern graph that is not well-formed and not commuting

– for all $p \in N_P^0$, $|N_G| = 1$;
– for all $p \in N_P^1$ and the unique $e \in E_G$, $N_G = \{\mathsf{src}_G(e), \mathsf{tgt}_G(e)\}$ and $p \lhd_P$ together cover N_G; and
– for all $p \in N_P^{2+}$, $p \lhd_P$ together cover G.

In words, the patterns of level-0 pattern nodes consist of a single node, the patterns of level-1 pattern nodes consist of a single edge and its end nodes, and the patterns on any other level are determined by the combined images of their predecessors. Another consequence of pattern morphisms being non-surjective is that, on well-formed pattern graphs, there are no pattern edges between nodes of the same layer. The universe of (well-formed) pattern graphs is denoted PGraph. Note that the pattern graph in Figure 2 is not well-formed: pattern nodes p_5, p_{12}, and p_{17} are not sufficiently covered by the incoming morphisms.

Definition 10 (pattern graph morphism). *A* pattern graph morphism *between pattern graphs $P, Q \in \mathsf{PGraph}$ is a graph morphism $m : P \to Q$, where,*
1. *for all $p \in N_P$, there exists an isomorphism $\varphi_p : \mathsf{lab}_P(p) \to \mathsf{lab}_Q(m(p))$; and*
2. *for all $d = \langle p, f, q \rangle \in E_P$, $\varphi_q \circ f = f' \circ \varphi_p$, with $f' = \mathsf{lab}_Q(m(d))$.*

Moreover, m is called closed *if, in addition,*
3. *for all $p \in N_P$ and $d' \in m(p)\lhd_Q$, there exists $d \in p\lhd_P$ with $m(d) = d'$; and*
4. *for all $N' \subseteq N_P$ and jointly surjective $\{d'_k \in m(p)\rhd_Q \mid p \in N'\}_{k \in K}$ (where K is some index set), there are jointly surjective $\{d_k \in p\rhd_P \mid p \in N'\}_{k \in K}$ with $m(d_k) = d'_k$, for all $k \in K$.* ◀

The definition above states that m maps pattern nodes with isomorphic patterns (condition 1) and that m is compatible with the simple graph morphisms of pattern edges modulo isomorphism (condition 2). Closedness indirectly imposes conditions on P: every pattern edge in Q whose target pattern node is in the morphism image should itself also be the image of some pattern edge in P (condition 3), and so should every jointly surjective set of pattern edges in Q whose source pattern nodes are in the morphism image (condition 4).

Definition 11. *Let P be a pattern graph.*

– *P is called* commuting *when for all $q \in N_P^{2+}$ and any distinct $d_1, d_2 \in q \triangleleft_P$, if $G = \mathsf{img}_P(d_1) \cap \mathsf{img}_P(d_2)$ is not an empty graph, then there exist $p \in N_P$ and parallel paths π_1, π_2 in P such that $\mathsf{lab}_P(p) = G$, $\mathsf{src}(\pi_i) = p$, $\mathsf{tgt}(\pi_i) = q$, and $d_i \in \pi_i$, for $i = 1, 2$.*

– *P is called* concrete *if it satisfies the following properties:*
  1. *for all distinct $p, q \in N_P$, $\mathsf{lab}_P(p) \neq \mathsf{lab}_P(q)$; and*
  2. *for all $\langle p, f, q \rangle \in E_P$, $f = \mathsf{emb}(\mathsf{lab}_P(p), \mathsf{lab}_P(q))$.* ◀

The commutativity condition states that common simple nodes and edges in patterns always stem from a common ancestor. The pattern graph in Figure 2 is not commuting: the pattern associated with pattern node p_{16} cannot be constructed from its predecessors since there is no common ancestor for simple node v_2.

The first concrete pattern graph condition states that all patterns are distinct and the second condition that identities of simple graph elements are preserved along pattern graph edges. For concrete pattern graphs P, we define the *flattening* of P as $\mathsf{flat}(P) = \bigcup_{p \in N_P} \mathsf{lab}_P(p)$. The following states that we can essentially always treat commuting pattern graphs as concrete.

Proposition 12. *Let P be a pattern graph.*
1. *If P is concrete, then P is commuting.*
2. *If P is commuting, there is a concrete pattern graph Q isomorphic to P.* ◀

So far we have not restricted the patterns occurring in a pattern graph, but in practice we will only use *typed* patterns.

Definition 13 (pattern type graph). *A* pattern type graph *T is a pattern graph such that $\mathsf{lab}_T(p) \not\simeq \mathsf{lab}_T(q)$ for all distinct $p, q \in N_T$. A T*-type morphism *is a closed pattern graph morphism to T.* ◀

Figure 3 shows an example of a pattern type graph. We call P a *T-pattern graph* if it is typable by T, *i.e.*, has a T-type morphism. It is easy to see that for a given pattern type graph T, any pattern graph P has at most one morphism to T. If this morphism exists but is not a type morphism (*i.e.*, is not closed), it is always possible to extend P to a T-pattern graph Q, namely by adding the elements required by the morphism closure conditions.

Proposition 14. *Let P be a concrete pattern graph and T a pattern type graph. If there exists a morphism $m : P \to T$, then there exists an unique (modulo isomorphism) concrete T-pattern graph $Q \supseteq P$, and $\mathsf{flat}(P) = \mathsf{flat}(Q)$.* ◀

We call Q the *closure* of P with respect to T, and denote it $\mathsf{close}_T(P)$. Given a pattern type graph T we can define the *lifting* operation $P = \mathsf{lift}_T(G)$ from simple graphs G to concrete T-pattern graphs P with typing morphism t:

– For all $H \subseteq G$ such that there exists an isomorphism $\varphi_H : H \to \mathsf{lab}_T(p')$ for some $p' \in N_T$, add a fresh p_H to N_P and let $\mathsf{lab}_P : p_H \mapsto H$, and $t : p_H \mapsto p'$.

– For all $p_H \in N_P$ and for every $d' \in t(p_H) \triangleleft_T$, let $F \subseteq H$ be defined by $F = \varphi_H^{-1}(\mathsf{img}_T(d'))$. Note that this implies $F \simeq \mathsf{lab}_T(\mathsf{src}_T(d'))$, and therefore

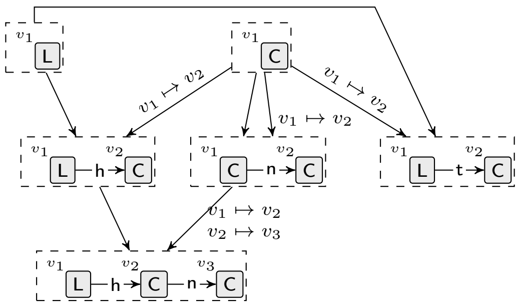

Fig. 3. Example of a pattern type graph

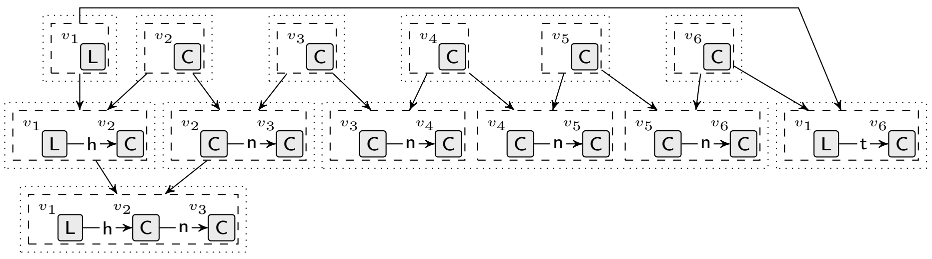

Fig. 4. Concrete pattern graph lifted from the simple graph of Figure 1 according to the pattern type graph of Figure 3. The pattern equivalence relation $\equiv$ (Definition 25) over nodes of the pattern graph is shown with dotted rectangles.

there exists a $p_F \in N_P$. Add a fresh edge d to E_P and let $\mathsf{src}_P : d \mapsto p_F$, $\mathsf{tgt}_P : d \mapsto p_H$, $\mathsf{lab}_P : d \mapsto \mathsf{emb}(F, H)$, and $t : d \mapsto d'$.

An example of a lifted concrete T-pattern graph is shown in Figure 4.

Proposition 15. *Let T be a pattern type graph.*

1. *For any simple graph G, $\mathsf{lift}_T(G)$ is a concrete T-pattern graph.*
2. *For any concrete T-pattern graph P, $\mathsf{lift}_T(\mathsf{flat}(P)) \simeq P$.* ◀

Concrete T-pattern graphs are transformed by lifting simple graph rules to pattern graph equivalents. Usually we only consider simple rules whose left hand sides are patterns in T, *i.e.*, for any simple graph rule $r = \langle L, R \rangle$, $L \simeq \mathsf{lab}_T(p)$ for some $p \in N_T$. In this sense, a set of simple rules $\mathcal{R}$ constrains the choice of type graph T, or, equivalently, T is extracted from set $\mathcal{R}$.

Definition 16 (pattern graph transformation rule). *A* pattern graph rule *$r = \langle \lambda, \rho \rangle$ consists of two concrete T-pattern graphs λ (the* LHS*) and ρ (the* RHS*), where for any $x_1 \in \lambda$ and $x_2 \in \rho$, $x_1 = x_2$ if and only if $\mathsf{lab}_\lambda(x_1) = \mathsf{lab}_\rho(x_2)$.* ◀

The definition above implies that there exists an identity pattern graph morphism on λ ensuring the equality of patterns. Based on this identity, we distinguish the sets of pattern graph elements deleted and created by pattern graph

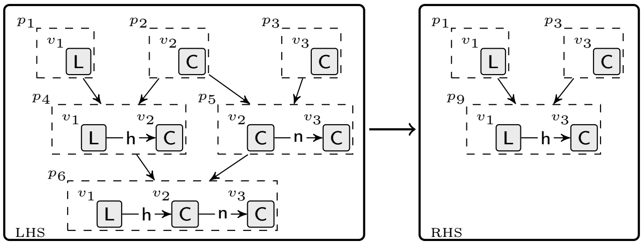

Fig. 5. Pattern graph rule equivalent to the get rule of Figure 1

rules, as done previously for simple graphs. Also, we use $\Upsilon = \lambda \cup \rho$ to denote the pattern graph resulting from the union of LHS and RHS, and Υ^{new} to represent the set of pattern graph elements created by the rule. We use PRule to denote the universe of pattern graph transformation rules, with one shown in Figure 5. A simple graph rule $r = \langle L, R\rangle$ and a pattern graph rule $r' = \langle \lambda, \rho\rangle$ are *equivalent* if $\mathsf{flat}(\lambda) = L$ and $\mathsf{flat}(\rho) = R$.

Essentially, what happens when a rule is applied is that graph elements are removed and others are added. In a concrete pattern graph, each simple graph element x is represented by a pattern node in N_P^0 (if x is a simple node) or N_P^1 (if x is a simple edge), and it also contributes to all successor patterns; so when x is removed, all those pattern nodes disappear. Conversely, adding simple graph elements to a pattern graph means adding new pattern nodes to N_P^0 and N_P^1, and then closing the resulting structure with respect to T.

Definition 17 (pattern graph transformation). *Let P be a concrete pattern graph, $r = \langle L, R\rangle$ a simple graph transformation rule, and $r' = \langle \lambda, \rho\rangle$ an equivalent pattern graph rule such that P and Υ^{new}, and $\mathsf{flat}(P)$ and U^{new} are disjoint. An* application *of r' into P involves finding a* match *μ of r' into P, which is an injective pattern graph morphism $\mu : \lambda \to P$. Match μ implicitly induces a simple graph match $m : L \to \mathsf{flat}(P)$. Extend μ to Υ and m to U by taking $\mu \cup \mathsf{id}_{\Upsilon^{\mathsf{new}}}$ and $m \cup \mathsf{id}_{U^{\mathsf{new}}}$, respectively, and let $N' = \{q \in N_P \mid p \in \mu(N^{\mathsf{del}}), p \leq_P q\}$ and $E' = \{d \in E_P \mid \mathsf{src}_P(d) \in N' or\ \mathsf{tgt}_P(d) \in N'\}$. Given such μ, r'* transforms *P into $\mathsf{close}_T(Q)$, where Q is defined by*

- $N_Q = (N_P \setminus N') \cup N^{\mathsf{new}}$ *and* $E_Q = (E_P \setminus E') \cup E^{\mathsf{new}}$;
- $\mathsf{src}_Q = (\mathsf{src}_P \cup (\mu \circ \mathsf{src}_\Upsilon))|_{E_Q}$ *and* $\mathsf{tgt}_Q = (\mathsf{tgt}_P \cup (\mu \circ \mathsf{tgt}_\Upsilon))|_{E_Q}$;
- *for all* $p \in N_Q$, $\mathsf{lab}_Q \colon \begin{cases} p \mapsto \mathsf{lab}_P(p) & \textit{if } p \notin N^{\mathsf{new}}, \\ p \mapsto m(\mathsf{lab}_\rho(p)) & \textit{otherwise; and} \end{cases}$
- *for all* $d \in E_Q$, $\mathsf{lab}_Q \colon d \mapsto \mathsf{emb}(\ \mathsf{lab}_Q(\mathsf{src}_Q(d)),\ \mathsf{lab}_Q(\mathsf{tgt}_Q(d))\ )$. ◀

As with simple graph transformations, we can satisfy the disjointness assumptions of the definition above by taking isomorphic copies of r and r' and the result of the transformation is the same, modulo isomorphism. It is easy to see that Q is a concrete pattern graph and therefore its closure w.r.t. T is well-defined. Figure 6 shows an example of a pattern graph transformation.

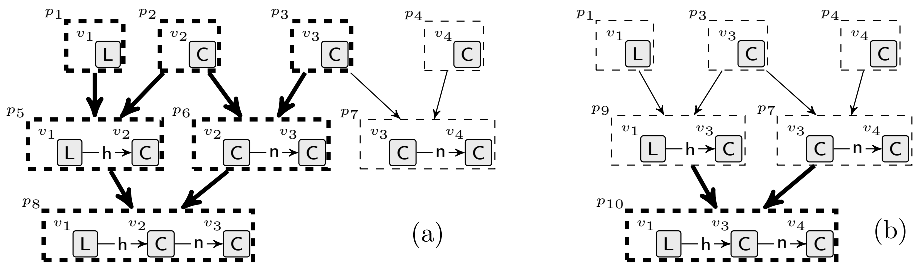

Fig. 6. Example of a pattern graph transformation. (a) Pattern graph to be transformed, with match of rule get shown in bold. (b) Resulting pattern graph, with elements added by the closure operation shown in bold.

We come now to the first major result of this paper: simple and pattern graph transformations are equivalent.

Theorem 18 (transformation equivalence). *Let G be a simple graph, $r = \langle L, R\rangle$ a simple graph rule, and $r' = \langle \lambda, \rho\rangle$ an equivalent pattern graph rule.*

1. *If $G \xrightarrow{r} H$ is a simple graph transformation then there is a pattern graph transformation $\mathsf{lift}_T(G) \xRightarrow{r'} Q$ with $Q \simeq \mathsf{lift}_T(H)$.*
2. *If $\mathsf{lift}_T(G) \xRightarrow{r'} Q$ is a pattern graph transformation then there is a simple graph transformation $G \xrightarrow{r} H$ with $Q \simeq \mathsf{lift}_T(H)$.* ◀

The equivalence between simple graph and pattern graph transformations is used to show the equivalence between simple graph transition systems (SGTS, Definition 8) and pattern graph transition systems (PGTS, defined below).

Definition 19 (pattern graph grammar). *A* pattern graph grammar $\mathcal{P}_T = \langle \mathcal{R}_T, P_0\rangle$ *has a set of pattern graph rules $\mathcal{R}_T$ and an initial T-pattern graph P_0.* ◀

We say that a simple graph grammar $\mathcal{G} = \langle \mathcal{R}, G_0\rangle$ and a pattern graph grammar $\mathcal{P}_T = \langle \mathcal{R}'_T, P_0\rangle$ are *equivalent* if for any simple graph rule $r \in \mathcal{R}$ there exists an equivalent pattern graph rule $r' \in \mathcal{R}'_T$, and vice versa; and $P_0 = \mathsf{lift}_T(G_0)$.

Definition 20 (PGTS). *A* pattern graph transition system $PGTS = \langle \mathcal{S}, \Rightarrow, \iota\rangle$ *consists of a set of states $\mathcal{S} \subseteq \mathsf{PGraph}$, a set of rule applications $\Rightarrow\ \subseteq \mathcal{S} \times \mathsf{PRule} \times \mathcal{S}$, and an initial state $\iota \in \mathcal{S}$. Grammar $\mathcal{P}_T$* generates *a $PGTS_{\mathcal{P}}$ if $\iota = P_0$ and $\mathcal{S}$ is the minimal set of graphs such that $P \in \mathcal{S}$ and $P \xRightarrow{r} Q$ for $r \in \mathcal{R}_T$ implies that there exists $Q' \in \mathcal{S}$ such that $Q \simeq Q'$ and $P \xRightarrow{r} Q'$ is a transition.* ◀

We conclude this section with our second major result, which establishes the relation between a SGTS and a PGTS generated by equivalent grammars.

Theorem 21. *Let T be a pattern type graph, $\mathcal{G}$ a simple graph grammar and $\mathcal{P}_T$ a pattern graph grammar equivalent to $\mathcal{G}$. Transition systems $SGTS_{\mathcal{G}}$ and $PGTS_{\mathcal{P}}$ are isomorphic.* ◀

Isomorphism is a quite interesting result because it implies that satisfaction of μ-calculus formulae (and thus also CTL*, CTL, and LTL formulae) are preserved among the two systems. This in turn means that we can discard the SGTS and perform verification (model-checking) on the pattern graph level. However, a PGTS may still be infinite, effectively preventing its construction.

4 Pattern Shapes

Pattern graphs are abstracted into pattern shapes. As usual with structural abstraction, equivalent structures (patterns) are collapsed into an abstract representative, while keeping an approximate count of the number of concrete elements collapsed. We use ω to denote an upper bound on the set of natural numbers and we write $\mathbb{N}^\omega = \mathbb{N} \cup \{\omega\}$. A *multiplicity* is an element of set $\mathcal{M} = \{\langle i, j\rangle \in (\mathbb{N} \times \mathbb{N}^\omega) \mid i \leq j\}$ that is used to represent an interval of consecutive values taken from $\mathbb{N}^\omega$. Given $\langle i, j\rangle \in \mathcal{M}$, if $i = j$ we write it as $\mathbf{i}$ and if $j = \omega$, we use $\mathbf{i}^+$ as short-hand. Multiplicity $\mathbf{1}$ is called *concrete*. Set $\mathcal{M}$ is infinite, since i and j are taken from infinite sets. To ensure finiteness, we need to define a bound of precision, which limits the possible values of i and j.

Definition 22 (bounded multiplicity). *A* bounded multiplicity *is an element of set $\mathcal{M}^\mathbf{b} \subset \mathcal{M}$, defined, for a given bound $\mathbf{b} \in \mathbb{N}$, as $\mathcal{M}^\mathbf{b} = \{\langle i, j\rangle \in \mathcal{M} \mid i \leq \mathbf{b} + 1,\ j \in \{0, \ldots, \mathbf{b}, \omega\}\}$.* ◀

It is straightforward to define arithmetic operations and relations over multiplicities. For this paper, it suffices to consider *(i)* the *subsumption* relation $\sqsubseteq$, defined as $\langle i, j\rangle \sqsubseteq \langle i', j'\rangle$ if $i \geq i'$ and $j \leq j'$, and *(ii)* relation $\leq$, defined as $\langle i, j\rangle \leq \langle i', j'\rangle$ if $j \leq j'$. Also, it is simple to define a function $\beta^\mathbf{b} : \mathcal{M} \to \mathcal{M}^\mathbf{b}$ that approximates multiplicities according to a bound $\mathbf{b}$. Let $M \subset \mathcal{M}$ be a set of multiplicities. We write $\sum^\mathbf{b} M$ to denote the bounded multiplicity sum over elements of M, as a short-hand notation for $\beta^\mathbf{b}(\sum M)$. From here on we assume the existence of two bounds, $\mathbf{n}, \mathbf{e} \in \mathbb{N}$, called *node* and *edge* bounds, respectively.

Definition 23 (pattern shape). *A* pattern shape *S is a pattern graph with additional node and edge multiplicity functions, denoted $\mathsf{mult}^\mathsf{n}_S : N_S \to \mathcal{M}^\mathbf{n}$ and $\mathsf{mult}^\mathsf{e}_S : E_S \to \mathcal{M}^\mathbf{e}$, respectively.* ◀

Function $\mathsf{mult}^\mathsf{n}_S$ indicates how many concrete patterns were folded into an abstract pattern node, up to bound $\mathbf{n}$. Function $\mathsf{mult}^\mathsf{e}_S$, on the other hand, counts locally, *i.e.*, it indicates how many edges of a certain type *each* of the concrete nodes had, up to bound $\mathbf{e}$. We write PShape to denote the universe of pattern shapes and we consider $\mathsf{mult}_S = \mathsf{mult}^\mathsf{n}_S \cup \mathsf{mult}^\mathsf{e}_S$. Pattern graphs can be *trivially extended* to pattern shapes by associating multiplicity maps according to the kind of pattern graph. For a pattern type graph T we associate the most abstract multiplicity to all elements of T, *i.e.*, $\mathsf{mult}_T(x) \mapsto \mathbf{0}^+$, for all $x \in T$. For any other pattern graph P, its trivial extension is obtained by making $\mathsf{mult}_P(x) \mapsto \mathbf{1}$, for all $x \in P$. From here on, we consider that trivial extensions of pattern graphs are taken when necessary. The distinct choice for multiplicities in pattern type graphs is motivated by the definition below.

Definition 24 ($\preceq$-morphism). *A* $\preceq$-morphism *between pattern shapes $X, Y \in \mathsf{PShape}$ is a pattern graph morphism $m : X \to Y$ that relates multiplicities according to relation $\preceq$,* i.e.,

- *for all $p' \in N_Y$, $\sum^{\mathbf{n}}_{p \in m^{-1}(p')} \mathsf{mult}^{\mathsf{n}}_X(p) \preceq \mathsf{mult}^{\mathsf{n}}_Y(p')$; and*
- *for all $d' \in E_Y$ and all $p \in N_X$, $\sum^{\mathbf{e}}_{d \in C} \mathsf{mult}^{\mathsf{e}}_X(d) \preceq \mathsf{mult}^{\mathsf{e}}_Y(d')$, where $C = m^{-1}(d') \cap p \rhd_X$.*

A pattern shape morphism *is defined to be a $\leq$-morphism, and a $\sqsubseteq$-morphism is called a* subsumption morphism*.* ◀

We write $\mathsf{depth}(S)$ to denote the maximum layer of pattern shape S that is not empty, *i.e.*, $\mathsf{depth}(S) = i \in \mathbb{N}$ such that $|N^i_S| \neq 0$ and for all $j > i$, $|N^j_S| = 0$. Let A be a set and $\equiv \subseteq A \times A$ be an equivalence relation over A. For $x \in A$, we write $[x]_\equiv$ to denote the equivalence class of x induced by $\equiv$, *i.e.*, $[x]_\equiv = \{y \in A \mid y \equiv x\}$ and we write $A/\!\equiv$ to denote the set of equivalence classes in A, *i.e.*, $A/\!\equiv\, = \{[x]_\equiv \mid x \in A\}$. For any $C_1, C_2 \subseteq A$, $C_1 \equiv C_2$ if for all $x_1 \in C_1$ there exists $x_2 \in C_2$ such that $x_1 \equiv x_2$, and vice versa.

Definition 25 (pattern equivalence). *Let S be a T-pattern shape and $t : S \to T$ be the typing morphism. The* pattern equivalence $\equiv$ *is defined as the smallest symmetrical relation over $N_S \times N_S$ and $E_S \times E_S$ where*

- *for any $p_1, p_2 \in N_S$, $p_1 \equiv p_2$ if $t(p_1) = t(p_2)$ and for all $C_1 \in (p_1 \rhd_S)/\!\equiv$, there exists $C_2 \in (p_2 \rhd_S)/\!\equiv$ such that $C_1 \equiv C_2$ and $\sum^{\mathbf{e}}_{d_1 \in C_1} \mathsf{mult}^{\mathsf{e}}_S(d_1) = \sum^{\mathbf{e}}_{d_2 \in C_2} \mathsf{mult}^{\mathsf{e}}_S(d_2)$; and*
- *for any $d_1, d_2 \in E_S$, $d_1 \equiv d_2$ if $t(d_1) = t(d_2)$ and $\mathsf{tgt}_S(d_1) \equiv \mathsf{tgt}_S(d_2)$.* ◀

The definition above implies that only nodes of the same layer can be equivalent, and that equivalent nodes have the same number of outgoing edges of each type into the same classes. Also, note that the second condition for node equivalence is vacuously true for nodes in layer $\mathsf{depth}(S)$, which gives a base case for the inductive definition. Given $\equiv$ we can derive a finer relation $\triangleq$ that groups edges per source equivalence classes: $d_1 \triangleq d_2$ if $d_1 \equiv d_2$ and $\mathsf{src}_S(d_1) \equiv \mathsf{src}_S(d_2)$.

Definition 26 (canonical pattern shape). *Let X be a T-pattern shape and let $t : X \to T$ be the typing morphism. The* canonical pattern shape *of X w.r.t. equivalence relation $\equiv$ is the pattern shape Y, where $N_Y = N_X/\!\equiv$ and $E_Y = E_X/\!\triangleq$, and for all $[p]_\equiv \in N_Y$, $p \in N_X$, $[d]_\triangleq \in E_Y$ and $d \in E_X$:*

- $\mathsf{src}_Y : [d]_\triangleq \mapsto [\mathsf{src}_X(d)]_\equiv$ *and* $\mathsf{tgt}_Y : [d]_\triangleq \mapsto [\mathsf{tgt}_X(d)]_\equiv$*;*
- $\mathsf{lab}_Y : [p]_\equiv \mapsto \mathsf{lab}_T(t(p))$ *and* $\mathsf{lab}_Y : [d]_\triangleq \mapsto \mathsf{lab}_T(t(d))$*;*
- $\mathsf{mult}^{\mathsf{n}}_Y : [p]_\equiv \mapsto \sum^{\mathbf{n}}_{p' \in [p]_\equiv} \mathsf{mult}^{\mathsf{n}}_X(p')$*; and*
- $\mathsf{mult}^{\mathsf{e}}_Y : [d]_\triangleq \mapsto \sum^{\mathbf{e}}_{d' \in C} \mathsf{mult}^{\mathsf{e}}_X(d')$*, where $C = (\mathsf{src}_X(d) \rhd_X) \cap [d]_\triangleq$.* ◀

In words, nodes and edges of canonical pattern shape Y are the equivalence classes of X, the labelling of Y takes the type associated with each equivalence class of X, and the multiplicities of Y are the bounded sum of the multiplicities of elements in the equivalence classes of X. Let P be a pattern graph and S be a pattern shape. We write $\mathsf{abstract}(P)$ and $\mathsf{normalise}(S)$ to denote the canonical pattern shape of P and S, respectively. Morphism $\alpha : P \to \mathsf{abstract}(P)$ is called

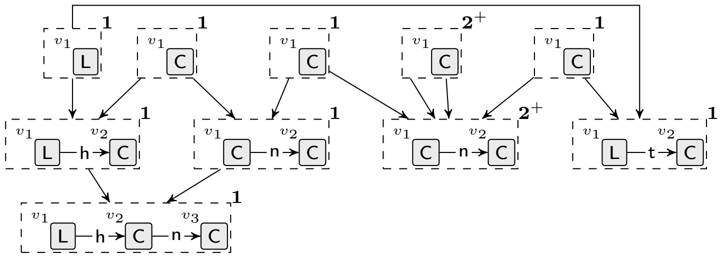

Fig. 7. Canonical pattern shape obtained when considering the pattern graph and equivalence relation of Figure 4. Pattern edge morphisms are not explicitly shown. Node multiplicities are given at the upper right corner of each pattern node. All edge multiplicities are **1** and are not shown.

an *abstraction morphism* and morphism $\Omega : S \rightarrow \mathsf{normalise}(S)$ is called a *normalisation morphism*. Both α and Ω are instances of subsumption morphisms.

Figure 7 shows an example of a canonical pattern shape. We use $\mathsf{CanPShape}_T^{\mathbf{n},\mathbf{e}}$ to denote the universe of canonical shapes typable by T and bounded by **n** and **e**. Our third major result follows, establishing finiteness of the abstract state space.

Theorem 27. *Given a pattern type graph T and bounds $\mathbf{n}, \mathbf{e} \in \mathbb{N}$, universe $\mathsf{CanPShape}_T^{\mathbf{n},\mathbf{e}}$ is finite (under isomorphism).* ◀

We now proceed to define how pattern shapes can be transformed by rules. Given a pattern shape S and a set of pattern nodes $N' \subseteq N_S$, let $N'_\square = \{q \in N_S \mid \exists o \in N_S, p \in N' : o \leq_S p,\ o \leq_S q\}$ and $E'_\square = \{d \in E_S \mid \mathsf{src}_S(d) \in N'_\square\}$. Pair $\mathsf{env}_S(N') = \langle N'_\square, E'_\square \rangle$ is called the *environment* of N' in S, *i.e.*, the pattern graph elements that can be affected by a pattern graph transformation matched on N'. Environment $\mathsf{env}_S(N')$ can be trivially turned into a sub-graph of S.

Definition 28 (rule pre-match/match into pattern shapes). *Let S be a T-pattern shape and $r = \langle \lambda, \rho \rangle$ be a pattern graph rule. A* pre-match *of r into S is a pattern shape morphism $\mu : \lambda \rightarrow S$. We call μ a* match *if $\mathsf{env}_S(\mu(N_\lambda))$ is a concrete pattern graph and for all $x \in \mathsf{env}_S(\mu(N_\lambda))$, $\mathsf{mult}_S(x) = \mathbf{1}$.* ◀

Given a match, a *concrete pattern shape transformation* proceeds as a pattern graph transformation on the environment sub-graph.

Definition 29 (concrete pattern shape transformation). *Let X be a T-pattern shape, $r = \langle \lambda, \rho \rangle$ be a pattern graph rule, $\mu : \lambda \rightarrow X$ be a match of r into X, $X_\square = \mathsf{env}_X(\mu(N_\lambda)) \subseteq X$ be the environment sub-graph of $\mu(N_\lambda)$ in X, and let $X_\square \overset{r}{\Longrightarrow} Y'$ be a pattern graph transformation. The result of a* concrete pattern shape transformation *of X is the T-pattern shape Y, where*
- $N_Y = (N_X \setminus N_{X_\square}) \cup N_{Y'}$ *and* $E_Y = \{d \in E_X \mid \mathsf{src}_X(d), \mathsf{tgt}_X(d) \in N_Y\} \cup E_{Y'}$;
- $\mathsf{src}_Y = (\mathsf{src}_X \cup \mathsf{src}_{Y'})|_{E_Y}$ *and* $\mathsf{tgt}_Y = (\mathsf{tgt}_X \cup \mathsf{tgt}_{Y'})|_{E_Y}$;
- $\mathsf{lab}_Y = (\mathsf{lab}_X \cup \mathsf{lab}_{Y'})|_{(N_Y \cup E_Y)}$; *and*
- $\mathsf{mult}^{\mathsf{n}}_Y = (\mathsf{mult}^{\mathsf{n}}_X \cup \mathsf{mult}^{\mathsf{n}}_{Y'})|_{N_Y}$ *and* $\mathsf{mult}^{\mathsf{e}}_Y = (\mathsf{mult}^{\mathsf{e}}_X \cup \mathsf{mult}^{\mathsf{e}}_{Y'})|_{E_Y}$. ◀

We write $X \overset{r}{\Longrightarrow} Y$ to denote a concrete pattern shape transformation, which we can now use to define transformations for canonical pattern shapes.

Definition 30 (canonical pattern shape transformation). *Given a canonical T-pattern shape X, a rule $r = \langle \lambda, \rho \rangle$, and a pre-match $\mu : \lambda \to X$, let X' be a T-pattern shape such that $\Omega_X : X' \to X$ is a normalisation morphism, $\mu' : \lambda \to X'$ is a match of r into X', and $\mu = \Omega_X \circ \mu'$. The* canonical pattern shape transformation *of X is the canonical T-pattern shape Y, where $X' \overset{r}{\Longrightarrow} Y'$ is a concrete pattern shape transformation and $\Omega_Y : Y' \to Y$ is a normalisation morphism.* ◀

Pattern shape X' is called a *materialisation* of X according to pre-match μ. An essential property is that a materialisation always exists, for any pre-match. We write $X \Rrightarrow^r Y$ to denote a canonical pattern shape transformation. Similarly to what was done with simple graphs and pattern graphs, rule applications on pattern shapes produce a *pattern shape transition system* (PSTS).

Definition 31 (PSTS). *A* pattern shape transition system $PSTS = \langle \mathcal{S}, \Rrightarrow, \iota \rangle$ *consists of a set of states $\mathcal{S} \subseteq \mathsf{CanPShape}_T^{\mathbf{n},\mathbf{e}}$, a set of rule applications $\Rrightarrow \subseteq \mathcal{S} \times \mathsf{PRule} \times \mathcal{S}$, and an initial state $\iota \in \mathcal{S}$. A pattern graph grammar $\mathcal{P}_T = \langle \mathcal{R}_T, P_0 \rangle$* generates *a $PSTS_{\mathcal{P}}$ if $\iota = \mathsf{abstract}(P_0)$ and $\mathcal{S}$ is the minimal set of canonical pattern shapes such that $X \in \mathcal{S}$ and $X \Rrightarrow^r Y$ for $r \in \mathcal{R}_T$ implies that there exists $Y' \in \mathcal{S}$ such that $Y \simeq Y'$ and $X \Rrightarrow^r Y'$ is a transition.* ◀

Our last result establishes the connection between concrete and abstract spaces.

Theorem 32. *Transition system $PSTS_{\mathcal{P}}$ simulates $PGTS_{\mathcal{P}}$.* ◀

An immediate consequence of PSTS being an over-approximation is that verification can then be carried out on the abstract domain, which is always finite. As usual with over-approximations, if a property holds at the pattern shape level then it is guaranteed to hold at the pattern graph level; however if a property is false for pattern shapes then we cannot be sure that it is also false for pattern graphs since the abstraction can introduce spurious behaviour [1]. The results at the pattern graph level transfer directly to simple graphs due to Theorem 21.

5 Related Work

Perhaps the most well-known method for the verification of infinite-state GT systems is the Petri graph unfolding technique, by König *et al.*, initially presented in [1]. Given a graph grammar this method extracts an *approximated unfolding*: a finite structure (called *Petri graph*) that is composed of a hyper-graph and a Petri net. The Petri graph captures all structure that can occur in the reachable graphs of the system, and dependencies for rule applications are recorded by net transitions. The structure obtained can then be used to check safety properties in the original system. If a spurious counter-example is introduced, the abstraction can be incrementally refined [10]. These techniques are implemented in the tool AUGUR which is now in its second version [11].

The approaches presented in [19,17] use a backwards reachability analysis for hyper-edge replacement grammars, where a search is started from a forbidden graph configuration and traverses backwards trying to reach an initial graph. This technique is applied to ad-hoc network routing and heap analysis, respectively, but is not guaranteed to terminate. In [5], an invariant checking method is developed for determining statically if a given forbidden pattern is an inductive invariant of a given rule. States that may lead to a forbidden graph are described symbolically, yielding a representation that is both complete and finite.

Our own take on GT abstractions was inspired by the seminal work on shape analysis by Sagiv *et al.* [18] and lead to theoretical results on *graph shapes* [13,14]. A similar approach called *partner abstraction* was developed in parallel by Bauer [2,4] and later these two approaches were unified in the framework of *neighbourhood abstraction* [3]. In [15,20] we describe the implementation effort for integrating neighbourhood abstraction into GROOVE [12,8], our GT tool set.

The main advantage of pattern abstraction over neighbourhood abstraction is the flexibility in specifying which structures should be collapsed. Precision of the neighbourhood method can be adjusted via a radius parameter but this radius is always the same when analysing the equivalence of nodes and edges. On the other hand, the pattern-based method is much more fine-grained: patterns of various sizes can be represented in the type graph, and are considered independently by the abstraction. Roughly speaking, a radius i neighbourhood abstraction can be simulated by a pattern abstraction using a type graph T with depth i, where all possible simple graphs of size smaller or equal to i occur as patterns in T.

6 Conclusions and Future Work

This paper presented a new method for a property-driven abstraction of graphs, based on a pre-defined collection of patterns of interest, represented as a pattern type graph. Pattern-based abstraction leads to a finite over-approximation of (infinite-state) graph transformation systems and as such the abstraction can be used for system verification. Furthermore, the technique lends itself nicely to abstraction refinement: one can start with a rather minimal type graph and then add more patterns to it to make the abstraction more precise when necessary.

The theory here presented, while sound, still leaves certain steps underspecified. In particular, the materialisation of canonical pattern shapes is of importance, since it may have a significant influence on the size of abstract state spaces. Based on our previous experience of implementing the theory of neighbourhood abstraction, we defer the definition of a materialisation algorithm to a later implementation phase, when a proper practical analysis of the trade-off between performance and abstraction precision can be made.

An interesting side-effect of developing a theory of pattern graph transformation is that structures used in incremental pattern matching like RETE networks [9] can now also be formalised as pattern graphs. To the best of our knowledge, this is the first time such structures were considered under a more formal focus.

We plan to implement this new theory into GROOVE, so that experiments can be carried out to gauge how suitable the proposed abstraction is in practice.

References

1. Baldan, P., Corradini, A., König, B.: A Static Analysis Technique for Graph Transformation Systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 381–395. Springer, Heidelberg (2001)
2. Bauer, J.: Analysis of Communication Topologies by Partner Abstraction. PhD thesis, Universität des Saarlandes (2006)
3. Bauer, J., Boneva, I., Kurban, M., Rensink, A.: A modal-logic based graph abstraction. In: [7]
4. Bauer, J., Wilhelm, R.: Static Analysis of Dynamic Communication Systems by Partner Abstraction. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 249–264. Springer, Heidelberg (2007)
5. Becker, B., Beyer, D., Giese, H., Klein, F., Schilling, D.: Symbolic invariant verification for systems with dynamic structural adaptation. In: ICSE. ACM (2006)
6. de Lara, J., Varro, D. (eds.): GraBaTs. ECEASST, vol. 32. EASST (2010)
7. Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.): ICGT 2008. LNCS, vol. 5214. Springer, Heidelberg (2008)
8. Ghamarian, A.H., de Mol, M., Rensink, A., Zambon, E., Zimakova, M.: Modelling and analysis using GROOVE. STTT 14(1) (2012)
9. Ghamarian, A.H., Rensink, A., Jalali, A.: Incremental pattern matching in graph-based state space exploration. In: [6]
10. König, B., Kozioura, V.: Counterexample-Guided Abstraction Refinement for the Analysis of Graph Transformation Systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 197–211. Springer, Heidelberg (2006)
11. König, B., Kozioura, V.: Augur 2 - a new version of a tool for the analysis of graph transformation systems. ENTCS 211 (2008)
12. Rensink, A.: The GROOVE Simulator: A Tool for State Space Generation. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 479–485. Springer, Heidelberg (2004)
13. Rensink, A.: Canonical Graph Shapes. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 401–415. Springer, Heidelberg (2004)
14. Rensink, A., Distefano, D.: Abstract graph transformation. In: Workshop on Software Verification and Validation (SVV). ENTCS, vol. 157 (2006)
15. Rensink, A., Zambon, E.: Neighbourhood abstraction in GROOVE. In: [6]
16. Rensink, A., Zambon, E.: Pattern-based graph abstraction (extended version). Technical report, University of Twente, Enschede, The Netherlands (2012)
17. Rieger, S., Noll, T.: Abstracting complex data structures by hyperedge replacement. In: [7]
18. Sagiv, S., Reps, T.W., Wilhelm, R.: Parametric shape analysis via 3-valued logic. ToPLaS 24(3) (2002)
19. Saksena, M., Wibling, O., Jonsson, B.: Graph Grammar Modeling and Verification of Ad Hoc Routing Protocols. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 18–32. Springer, Heidelberg (2008)
20. Zambon, E., Rensink, A.: Graph subsumption in abstract state space exploration. In: GRAPHITE (Pre-proceedings) (2012)