Abstract Graph Transformation

Arend Rensink
Department of Computer Science, University of Twente
rensink@cs.utwente.nl

Dino Distefano
Department of Computer Science, Queen Mary University of London
ddino@dcs.qmul.ac.uk

January 31, 2005

Abstract

Graphs may be used as representations of system states in operational semantics and model checking; in the latter context, they are being investigated as an alternative to bit vectors. The corresponding transitions are obtained as derivations from graph production rules.

In this paper we propose an abstraction technique in this framework: the state graphs are contracted by collecting nodes that are sufficiently similar (resulting in smaller states and a finite state space) and the application of the graph production rules is lifted to this abstract level. Since graph abstractions and rule applications can all be computed completely automatically, we believe that this can be the core of a practically feasible technique for software model checking.

1 Introduction

We study state-based models of system behaviour; our particular interest is in *software* systems. Our eventual aim is to develop tools to support the verification of software through such models. For this purpose, it is imperative that the models have an effective finite description. We propose to use *abstraction* as a means to obtain finite approximations of behavioral models. In this paper we describe a technique to define such approximations automatically for models consisting of *graphs*, with labelled edges over a finite alphabet, as states and *graph transformations* as transitions.

The abstract model we propose is strongly inspired by *shape graphs* as introduced in [19] and worked out further in [20]. The abstraction is based on *structural similarity* of nodes of the state graphs, described previously in [14] and shown there to give rise to a *finite* set of abstract graphs (called *shapes*, following the terminology of [19]). The contribution of the present paper is that we also show how to *transform* shapes, in such a way that all transitions between the concrete states (transformations of concrete graphs) give rise to transitions between the abstract states (transformations of shapes). Thus we have an over-approximation of the concrete transition system, on the basis of which we can make certain predictions about the actual system behaviour. Moreover, for every abstract transition there is at least one underlying concrete transition, meaning that we do not have spurious abstract transitions.

Motivation. We will us a running example of a circular buffer used to store data values. The buffer consists of an n-linked circular structure of C-nodes and a central B-node pointing to the (current) first and last cell through f- and l-edges. A cell can contain an object,

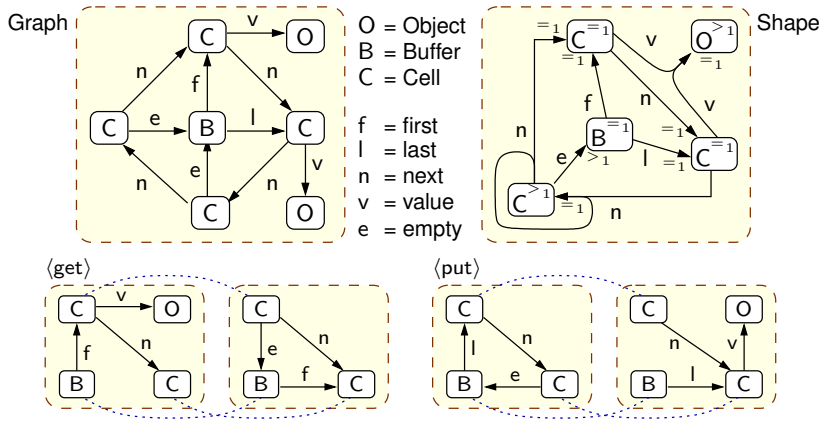

Figure 1: Example circular buffer with four cells, its shape, and two production rules

modelled by a v-edge to an O-node, or be empty, modelled by a e-edge back to the B-node. Fig. 1 shows an example buffer of four cells, two of which are empty. The shape of this buffer combines the (structurally similar) empty C-nodes and the O-nodes, and additionally specifies *multiplicities* on the nodes and incoming edges.[1] The $^{=}1$ on the incoming edge of the O-node, e.g., indicates that each concrete O-instance has exactly one incoming v-edge, which can come from *either* of the C-nodes.

To transform this example graph, Fig. 1 also shows two rules ⟨put⟩ and ⟨get⟩, each consisting of two graphs: a left hand side (LHS) and a right hand side (RHS). The rules describe the insertion and removal of objects, where for simplicity the nodes modelling the objects are actually created at insertion and deleted at removal. The effect of a rule is defined relative to a *matching* of the LHS, which is an injective graph morphism into the host graph. The images of those elements not in the RHS are subsequently removed from the host graph, whereas elements that are *fresh* in the RHS are added.

Given an initial graph and a set of production rules, we obtain a transition system by recursively applying all rules to all graphs. For instance, Fig. 2 shows the transition system for the graph and rules in Fig. 1. We propose to use such transition systems as the basis for model checking; first results are reported in [17]. However, for this technique to become practically feasible we need to address the following issues (among others):

- The models should be generic in the size of the data structures. As it is, for our example we get a different model if we start with a 5-cell buffer, etc.
- The models should be finite. As it is, if we add a rule to our example that may add a cell to a circular buffer when it is completely full, then the size of the graphs becomes unbounded and the state space becomes infinite.

By lifting graph transformations to the level of shapes we achieve both these goals. In fact, what we achieve is a completely automatic technique for state abstraction, in a setting where the models are inherently dynamic — that is, nodes and edges can be created and deleted at run-time. We believe that this is a promising basis for software verification, complementary to existing model checking techniques.

The abstract states will be *canonical* shapes, which is a sub-class satisfying some normalisation constraints. Their transformation is a three-step process.

Materialisation. This involves identifying the sub-shape where the rule applies (using the matching) and extracting an explicit, concrete copy of it. This is necessary to accurately mimic the effect of the transformation. The same principle can be found

[1] In this paper we assume that graphs are deterministic — defined below — which means that outgoing multiplicities are not needed. We write the edge multiplicities on the *opposite* end of the arrows than is usual in class diagrams.

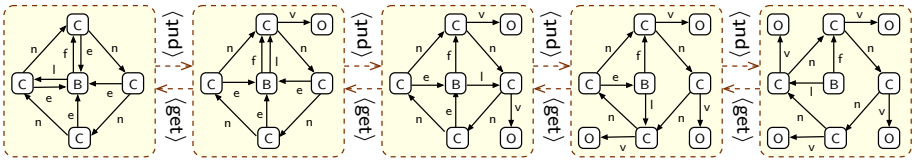

Figure 2: Concrete transition system of the circular buffer

in [19], from where we took the term "materialisation", but also in our own work [8, 7], where it is called "extraction."

Transformation. The transformation of a materialised shape is much like an ordinary graph transformation. We will show that this type of transformation both preserves and reflects transformations of the corresponding instance graphs.

Normalisation. The result of the transformation, although still an abstract graph, is typically outside the sub-class of *canonical* shapes. Therefore, we have to massage it to fit it back into that class. This may introduce additional non-determinism: an arbitrary shape typically gives rise to more than one canonical shape.

Structure of the paper. In Sect. 2 we define the basic notions of graphs and graph transformations, and we recall the shapes introduced in [14]. The materialisation, transformation and normalisation steps are described in Sections 3–5. In Sect. 6 we combine these steps and complete the framework. Finally, Sect. 7 summaries the paper and discusses related work. Proofs of all the theorems are included in App. A.

2 Definitions

2.1 Graphs and their transformations

In this section we define the basic graph formalism that we will use. In the following, L denotes a fixed, finite set of labels.

Definition 1 (graph and morphism) *A graph over L is a tuple $G = \langle N, E\rangle$ where N is a set of nodes and $E \subseteq N \times L \times N$ a set of labelled edges. G is called* deterministic *if $(v, a, w), (v, a, w') \in E$ implies $w = w'$.*

If $G = \langle N_G, E_G\rangle$ and $H = \langle N_H, G_H\rangle$ are graphs over L, a morphism *$\phi\colon G \to H$ is a function $\phi\colon N_G \to N_H$, extended to E_G by $\phi((v, a, w)) = (\phi(v), a, \phi(w))$, such that $\phi(E_G) \subseteq E_H$.*

An example deterministic graph was given in Fig. 1. Note that the node labels (B, C etc.) in that graph are actually not part of the formal definition; in fact they are superfluous (they can be derived from the edge labels), we have just included them for the sake of readability. In the following, $\mathbf{Gra}_L$ denotes the class of graphs and $\mathbf{DGra}_L$ the class of deterministic graphs. Given a edge $e = (v, a, w) \in E$ we call v the source, a the label and w the target of e. They are indicated as $src(e)$, $lab(e)$, and $tgt(e)$ respectively.

A bijective morphism $\phi : G \to H$ is called an *isomorphism* and two graphs G and H are called *isomorphic* (denoted $G \cong H$) if there exists an isomorphism between them.

In the following definitions, we present production rules and their applications in a purely constructive manner, instead of the algebraic characterisation found in the standard literature [3].

Definition 2 (production rule) *A* graph production rule *is a pair of graphs $P = (L, R)$ with $L, R \in \mathbf{DGra}_L$, called the* left hand side *(LHS) and* right hand side *(RHS), respectively. We also sometimes regard P itself as a single graph given by the union $L \cup R$, and we distinguish the following sets:*

- $N^{\mathsf{del}} = N_L \setminus N_R$ *and* $E^{\mathsf{del}} = E_L \setminus E_R$*, the elements to be deleted;*
- $N^{\mathsf{use}} = N_L \cap N_R$ *and* $E^{\mathsf{use}} = E_L \cap E_R$*, the elements used (but not changed);*
- $N^{\mathsf{new}} = N_R \setminus N_L$ *and* $E^{\mathsf{new}} = E_R \setminus E_L$*, the elements to be created.*

Two example production rules were given in Fig. 1. The set of production rules over L is denoted $\mathbf{Prod}_L$. The *application* of a production rule $P = (L, R)$ to a graph G entails finding a *matching* $m\colon L \to G$, which is an injective morphism from the LHS to the graph (also satisfying some other conditions, introduced below), and then removing from G the images of N^{del} and E^{del} and adding to the resulting graph the elements in N^{new} and E^{new}. Care must be taken, however, to ensure that the new elements are fresh and do not coincide with elements already in G. For this purpose, when discussing the application of a rule P to a graph G we will always assume P and G to be disjoint, i.e., $N_P \cap N_G = \emptyset$. This assumption can be satisfied without loss of generality by taking an isomorphic copy of P (and the result of the transformation does not depend on which isomorphic copy we take, modulo isomorphism).

Definition 3 (graph transformation) *Let* $P = (L, R) \in \mathbf{Prod}_L$ *and* $G \in \mathbf{Gra}_L$ *be disjoint. A* matching *for* P *in* G *is an injective morphism* $m\colon L \to G$ *such that the following conditions hold for all* $e \in E_G$*:*

1. *If* $\mathit{src}(e) \in m(N^{\mathsf{del}})$ *or* $\mathit{tgt}(e) \in m(N^{\mathsf{del}})$*, then* $e \in m(E^{\mathsf{del}})$*;*
2. *If* $\mathit{src}(e) \in m(N^{\mathsf{use}})$ *and* $\exists(m^{-1}(\mathit{src}(e)), \mathit{lab}(e), w) \in E^{\mathsf{new}}$*, then* $e \in m(E^{\mathsf{del}})$*.*

If m *is a matching for* P *in* G*, the transformation of* G *according to* P *and* m *is defined by* $((N_G \setminus m(N^{\mathsf{del}})) \cup N^{\mathsf{new}}, (E_G \setminus m(E^{\mathsf{del}})) \cup E^{\mathsf{new}})$*. We write* $G \xrightarrow{P,m} H$ *to denote that* m *is a matching for* P *in* G *and* H *is the resulting transformed graph.*

Application condition 1 is called the *dangling edge condition*; it is standard in the so-called *double pushout approach* to graph transformation (cf. [3]). Condition 2 could be called *preservation of determinism*; it is the most straightforward way to ensure that transformations remain in $\mathbf{DGra}$ (see Sect. 7 for a brief discussion). Example transformations (without the matchings) were shown in Fig. 2.

Proposition 4 *Let* $P \in \mathbf{Prod}_L$ *and* $G \in \mathbf{DGra}_L$*. If* $G \xrightarrow{P,m} H$ *then* $H \in \mathbf{DGra}_L$*.*

2.2 Multiplicities and Shapes

A multiplicity is an interval of natural numbers. Formally, we define the universe of multiplicities as $\mathbf{M} = \{(i, j) \in \mathbb{N} \times (\mathbb{N} \cup \{\star\}) \mid i \leq j\}$, where $\star$ is used to denote infinity (i.e., $i < \star$ for all $i \in \mathbb{N}$). We use μ to range over multiplicities. We write ${}^{=}i$ for (i, i), ${}^{>}i$ for $(i+1, \star)$ and ${}^{\geq}i$ for $(i, \star)$. The lower bound of a multiplicity $\mu \in \mathbf{M}$ is denoted by $\lfloor \mu \rfloor$ and the upper bound $\lceil \mu \rceil$; thus $\lfloor (i, j) \rfloor = i$ and $\lceil (i, j) \rceil = j$. Multiplicity μ is called positive if $\lfloor \mu \rfloor > 0$. We write $i \in \mu$ if $\lfloor \mu \rfloor \leq i \leq \lceil \mu \rceil$; based on this we define inclusion, $\mu_1 \subseteq \mu_2$, as $\forall i : i \in \mu_1 \Rightarrow i \in \mu_2$. A given set X *has multiplicity* μ, denoted $X{:}\mu$, if $|X| \in \mu$. The following defines two operations over multiplicities, where $\mu, \mu_1, \mu_2 \in \mathbf{M}$ and $i \in \mathbb{N}$ (note that $\star - i = \star + i = \star$ for all $i \in \mathbb{N}$):

$$\begin{aligned} \mu_1 + \mu_2 &= (\lfloor \mu_1 \rfloor + \lfloor \mu_2 \rfloor, \lceil \mu_1 \rceil + \lceil \mu_2 \rceil) \\ \mu - i &= (\max(0, \lfloor \mu \rfloor - i), \lceil \mu \rceil - i) \quad \text{if } \lceil \mu \rceil \geq i. \end{aligned}$$

The following expresses some algebraic properties of these various concepts.

Proposition 5 *Let* $\mu \in \mathbf{M}$*, and let* A, B *be arbitrary finite sets.*

1. *If* $A : \mu$ *then* $(A \setminus B) : \mu - |A \cap B|$*.*
2. *If* $i \leq \lceil \mu \rceil$ *then* $(\mu - i) + {}^{=}i \subseteq \mu$*.*

Multiplicities are used as basic ingredients for the definition of *shapes*. These are graphs where a multiplicity is associated with each node, stating how many concrete nodes it represents, and with each pair of node v and label a, stating how many incoming a-edges *each instance* of v has. Formally:

Definition 6 (shape) *A* shape *is a tuple* $S = \langle N, E, nd, in \rangle$ *with* $\langle N, E \rangle \in \mathbf{Gra}_L$ *(sometimes denoted by* G_S*), and*

- $nd : N \to \mathbf{M}$ *a node multiplicity function;*
- $in : N \to L \to \mathbf{M}$ *an incoming edge multiplicity function.*

S *is called* deterministic *if the following property holds:*

- *for all* $v \in N$ *such that* $nd(v) = {}^{=}1$ *and all* $a \in L$, $|\{w \mid (v, a, w) \in E\}| \leq 1$ *and* $|\{w \mid (w, a, v) \in E\}| \leq \lceil in(v)(a) \rceil$.

An example deterministic shape was shown in Fig. 1. We use $\mathbf{Sha}_L$ to denote the class of shapes over L, and $\mathbf{DSha}_L$ for the deterministic shapes. Each shape stands for a number of *instances*, which are concrete (deterministic) graphs. In this sense, a shape is comparable to a *type graph*; however, the multiplicities provide far more control over the structure of the instances. The relation between a shape and its instances is defined by the following notion of *shaping*.

Definition 7 (shaping) *Given a graph* $G \in \mathbf{DGra}_L$ *and a shape* $S \in \mathbf{Sha}_L$*, a* shaping *of* G *into* S *is a morphism* $s : G \to G_S$ *such that:*

1. *for all* $v \in N_S$, $s^{-1}(v) : nd(v)$*;*
2. *for all* $v \in N_G$ *and* $a \in L$, $\{w \in N_G \mid (w, a, v) \in E_G\} : in(s(v))(a)$*;*
3. *for all* $v \in N_G$ *and* $a \in L$*, if* $\exists (s(v), a, w) \in E_S$ *then* $\exists (v, a, w') \in E_G$.

We write $s : G \to S$ to denote that s is a shaping of G into S. It is important to note that, due to possible inconsistencies between multiplicity constraints, not all shapes have instances. If a shape admits instances we call it *consistent*. In [14] we have shown that the notion of consistency is decidable for arbitrary (finite) $S \in \mathbf{Sha}_L$.

A graph typically has (shapings into) many shapes; for instance, by changing the multiplicities of a shape into more permissive ones (i.e., that extend the old ones), all shapings remain preserved. In fact, shapes are interrelated by so-called *abstraction morphisms*.

Definition 8 (abstraction morphism) *Let* $S, T \in \mathbf{Sha}_L$*. An* abstraction morphism α *from* S *to* T *(written* $\alpha : S \to T$*) is a morphism* $\alpha : G_S \to G_T$ *such that:*

1. *for all* $v \in N_T$, $nd_T(v) \supseteq \sum nd_S(\alpha^{-1}(v))$*;*
2. *for all* $v \in N_S$ *and* $a \in L$, $in_T(\alpha(v))(a) \supseteq in_S(v)(a)$.*;*
3. *for all* $v \in N_S$ *and* $a \in L$, $\exists (\alpha(v), a, w) \in E_T$ *implies* $\exists (v, a, w') \in E_S$.

The following proposition states that (as expected) any instance of a shape is also an instance of a more abstract shape.

Proposition 9 *Let* $G \in \mathbf{DGra}_L$ *and* $S, T \in \mathbf{Sha}_L$*. If* s: $G \to S$ *is a shaping and* α: $S \to T$ *an abstraction, then* $\alpha \circ s$ *is a shaping of* G *into* T.

3 Materialisation

As discussed in the introduction, we will lift the application of graph production rules to shapes. We do this in two steps: first we *materialise* the shape, then we transform the materialised graph as if it were a concrete graph. Materialisation is done relative to a prospective matching of the rule's LHS. Since such a matching is not a shaping (the LHS is only a *fragment* of a graph and so the cardinality constraints in the shape are not necessarily met) we have to define first what kind of objects they are.

Definition 10 *Let $L \in \mathbf{DGra}_L$ and $S \in \mathbf{Sha}_L$. A* pre-shaping *p of L in S is a graph morphism $p\colon L \to G_S$ with the additional property that the* upper bounds *of the node and edge cardinalities are satisfied; i.e.,*

- *for all $v \in N_S$, $|p^{-1}(v)| \leq \lceil nd_S(v) \rceil$;*
- *for all $v \in N_G$ and $a \in L$, $|\{w \in N_G \mid (w, a, v) \in E_G\}| \leq \lceil in_S(p(v))(a) \rceil$.*

A pre-shaping p is called concrete *if the following additional properties hold:*

- *for all $v \in N_L$, $nd_S(p(v)) = {}^{=}1$;*
- *for all $(v, a, w) \in E_L$, $(p(v), a, w') \in E_S$ implies $w' = p(w)$.*

Pre-shapings extend injective morphisms from a graphs-to-graphs notion to a graphs-to-shapes notion. Concreteness means that the morphism maps only to nodes and edges that are uniquely identifiable in any concrete instance.

Proposition 11 *Let $L, G \in \mathbf{DGra}_L$ and $S \in \mathbf{Sha}_L$. If $f\colon L \to G$ is an injective morphism and $s\colon G \to S$ a shaping, then $s \circ f$ is a pre-shaping of L into S.*

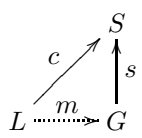

Figure 3: Visualisation of Prop. 12

The intuition is that the existence of a pre-shaping $p\colon L \to S$ indicates that L may be a *fragment* of an instance of S. We do not currently have a result that supports that intuition; that is, we do not know if or when the existence of p implies that there is an instance G with a (proper) shaping $s\colon G \to S$ and an embedding $m\colon L \to G$ such that $p = s \circ m$. We conjecture, however, that the results of [14] can easily be extended so as to reduce this property (for a given L and S) to an integer program, thus giving a decision procedure. For concrete pre-shapings, on the other hand, we have the following further property, depicted graphically in Fig. 3:

Proposition 12 *Let $L \in \mathbf{DGra}_L$ and $S \in \mathbf{Sha}_L$ and let $c\colon L \to S$ be a concrete pre-shaping. For any $G \in \mathbf{DGra}_L$ with a shaping $s\colon G \to S$, there is an injective morphism $m\colon L \to G$ such that $c = s \circ m$.*

Given a LHS L, a shape S and a pre-shaping $p\colon L \to S$, the *materialisation* of S relative to p is defined by disjointly adding a copy of L to S, connecting it to S where necessary, and adapting the node multiplicities of S to account for the extraction of one or more instances from them. W.l.o.g. we assume $N_L \cap N_S = \emptyset$; we define a function $\alpha_p\colon (N_L \cup N_S) \to N_S$ by

$$\alpha_p = p \cup id_S \ .$$

α_p is extended to edges as usual. The materialisation of S relative to p is defined by $S^{+p} = \langle N^{+p}, E^{+p}, nd^{+p}, in^{+p} \rangle$ with

$$\begin{array}{rcl} N^{+p} & = & N_L \cup N_S \\ E^{+p} & = & \alpha_p^{-1}(E_S) \setminus \{(v, a, w) \mid v \in N_L, \exists (v, a, w') \in E_L : w' \neq w\} \\ nd^{+p} : v & \mapsto & \begin{cases} nd_S(v) - |p^{-1}(v)| & \text{if } v \in N_S \\ {}^{=}1 & \text{otherwise} \end{cases} \\ in^{+p} : v & \mapsto & in_S(\alpha_p(v)) \ . \end{array}$$

An example materialisation is shown in Fig. 4. The first thing to show is the relation between L, S and S^{+p}. (See also Fig. 5.)

Proposition 13 *Let $L \in \mathbf{DGra}_L$ and $S \in \mathbf{Sha}_L$, and let $p\colon L \to S$ be a pre-shaping. α_p gives rise to an abstraction morphism from S^{+p} to S, and id_L gives rise to a concrete pre-shaping of L into S^{+p}, such that $p = \alpha_p \circ id_L$.*

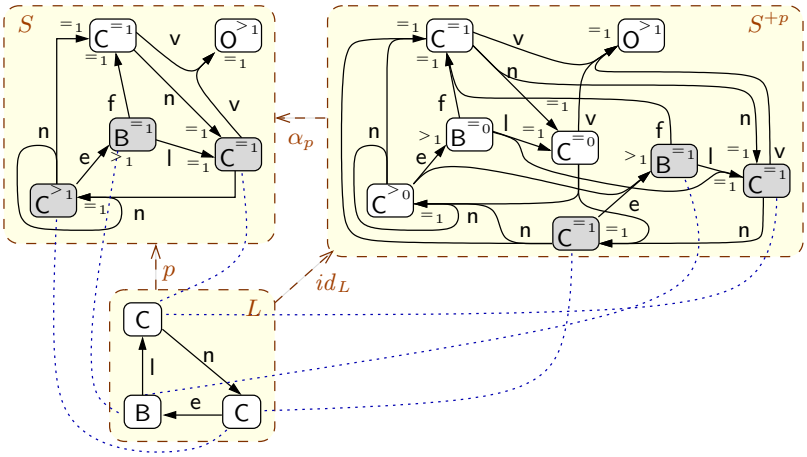

Figure 4: Materialisation of the shape in Fig. 1 w.r.t. the LHS of ⟨put⟩

The materialisation satisfies the following characteristic property (see Fig. 5):

Proposition 14 *Let $L, G \in \mathbf{DGra}_L$ and $S \in \mathbf{Sha}_L$. For an arbitrary injective morphism $m\colon L \to G$ and a shaping $s\colon G \to S$, let $p = s \circ m$; then there is a shaping $t\colon G \to S^{+p}$ with $s = \alpha_p \circ t$ and $t \circ m = id_L$.*

4 Transformation

In this section we prove the correctness of the abstraction we have defined, in the sense that a transformation of a shape with respect to a singular pre-shaping simulates a transformation of the underlying instance graphs and vice-versa.

First we extend the transformation definition from graphs (see Def. 3) to shapes.

Definition 15 (shape transformation) *Let $P = (L, R) \in \mathbf{Prod}_L$ and $S \in \mathbf{Sha}_L$ be disjoint. An* abstract matching *for P in S is a concrete pre-shaping $c\colon L \to S$ such that $c\colon L \to G_S$ is a (concrete) matching for P in the graph part of S. If c is an abstract matching for P in S, then the transformation of S according to P and s is defined by $T \in \mathbf{Sha}_L$ such that*

$$
\begin{aligned}
N_T &= (N_S \backslash c(N^{\mathsf{del}})) \cup N^{\mathsf{new}} \\
E_T &= (E_S \backslash c(E^{\mathsf{del}})) \cup E^{\mathsf{new}} \\
nd_T(v) &= \begin{cases} nd_S(v) & \text{if } v \in N_S \\ {}^{=}1 & \text{otherwise} \end{cases} \\
in_T(v)(a) &= \begin{cases} in_S(v)(a) - |\{w \mid (w,a,v) \in c(E^{\mathsf{del}})\}| \\ \qquad + {}^{=}|\{w \mid (w,a,v) \in E^{\mathsf{new}}\}| & \text{if } v \in N_S \\ {}^{=}|\{w \mid (w,a,v) \in E^{\mathsf{new}}\}| & \text{otherwise} \end{cases}
\end{aligned}
$$

We write $S \xrightarrow{P,c} T$ to denote that c is an abstract matching for P in S and T is the resulting transformed shape.

The following are two of the crucial theorems of this paper, providing the connection between abstract and concrete transitions.

Theorem 16 *Let $P = (L, R) \in \mathbf{Prod}_L$ and $S \in \mathbf{Sha}_L$, and assume $S \xrightarrow{P,c} T$. For any shaping $s\colon G \to S$, there exists a matching m for P in G such that $c = s \circ m$, and for $G \xrightarrow{P,m} H$ there is a shaping $t\colon H \to T$.*

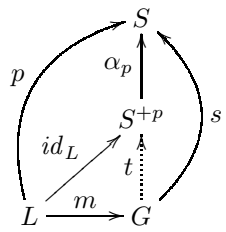

Figure 5: Visualisation of Propositions 13 and 14

Theorem 17 *Let $P = (L, R) \in \mathbf{Prod}_L$ and $G \in \mathbf{DGra}_L$, and assume $G \xrightarrow{P,m} H$. For any shaping $s\colon G \to S$ such that $c = s \circ m$ is concrete, and $S \xrightarrow{P,c} T$ with a shaping $t\colon H \to T$.*

5 Normalisation

The previous two sections have presented materialisation and transformation as two essential ingredients of abstract graph transformations. However, there is a third ingredient still missing for an effective technique: namely, we need to identify a *canonical abstraction level*, on which there exist only a finite number of shapes and to which the target graph of each transformation will be re-normalised. Failing this, due to materialisation the graphs under transformation will become ever larger and more concrete, so that the state space is still infinite and the advantages of abstraction are lost.

For this canonical abstraction level, we will rely on the ideas developed in [14, 16]. First of all, we select a collection of *base multiplicities* $\underline{\mathbf{M}} = \{^{=}0, ^{=}1, ^{>}1\}$ (chosen in such a way that every finite set has exactly one base multiplicity). $\underline{\mathbf{M}}^{>0} = \underline{\mathbf{M}} \setminus \{^{=}0\}$ denotes the set of *positive* base multiplicities. Next, we define the following notion of similarity $\sim_S \subseteq N_S \times N_S$ over nodes of a shape S:

$$v_1 \sim_S v_2 \;\Leftrightarrow\; in_S(v_1) = in_S(v_2) \wedge lab(src_S^{-1}(v_1)) = lab(src_S^{-1}(v_2)) \;. \qquad (1)$$

Hence, two nodes are similar if they have the same incoming edge multiplicities and outgoing edge labels.

Definition 18 (canonical shape) *A shape $S \in \mathbf{Sha}_L$ is called* canonical *if*

1. *S is deterministic;*
2. *for all $v \in N$, $nd(v) \in \underline{\mathbf{M}}^{>0}$;*
3. *for all $(v, a, w) \in E$, $in(v)(a) \in \underline{\mathbf{M}}^{>0}$;*
4. *for all $v, w \in N$, $v \sim_S w$ implies $v = w$.*

In words, a shape is canonical if it is deterministic, specifies positive base multiplicities for all nodes and edges (Clauses 2 and 3) and contains no non-trivially similar nodes (Clause 4).[2] The class of canonical shapes is denoted $\mathbf{CSha}_L$. An important fact from [14] is that $\mathbf{CSha}_L$ is finite for every finite set L.

We use the term *canonical* because, as we have shown in [16], there is an automatic way to obtain the *most concrete* canonical shape $can(G)$ of a given deterministic graph G. For an arbitrary shape S, on the other hand, there is typically not a *single* canonical shape that "covers" S in the sense of being more abstract (see Def. 8). Instead, we define a

[2] In [16] we required canonical shapes to be "fully satisfiable", meaning that there should exist a surjective shaping into them. The requirement of determinism is easier to maintain, but weaker than full satisfiability. As a consequence, in contrast to [16] it is not true that every graph has a unique canonical shaping.

function $norm$ such that $norm(S)$ is a *set* of canonical shapes, which is optimal in a sense (shown below).

To normalise multiplicities, we take all (non-empty) intersections of the multiplicities occurring in S with $\underline{\mathbf{M}}$. This is defined as follows (where $\mu \in \mathbf{M}$ and $f : X \to \mathbf{M}$):

$$\begin{array}{rcl} \mu/\underline{\mathbf{M}} & = & \{\mu' \in \underline{\mathbf{M}} \mid \exists i : i \in \mu \wedge i \in \mu'\} \\ f/\underline{\mathbf{M}} & = & \{g\colon X \to \underline{\mathbf{M}} \mid \forall x \in X : g(x) \in f(x)/\underline{\mathbf{M}}\} \ . \end{array}$$

The function $norm : \mathbf{Sha}_L \to \mathbf{2}^{\mathbf{CSha}_L}$ is then defined as follows:

$$norm : S \mapsto \{part(T) \mid T \in \mathbf{DSha}_L, T \lhd S, T \text{ consistent}\} \ .$$

where the property $T \lhd S$ is defined as the conjunction of the following conditions:

$$\begin{array}{lcl} N_T & \subseteq & \{(v,f) \mid v \in N_S,\ f \in in_S(v)/\underline{\mathbf{M}}\} \\ E_T & \subseteq & \{((v,f),a,(w,g)) \mid (v,a,w) \in E_S,\ g(a) \neq {}^{=}0\} \\ nd_T & \in & \{h\colon N_T \to \underline{\mathbf{M}}^{>0} \mid \forall v \in N_S : nd_S(v) \subseteq \sum_{(v,f)\in N_T} h((v,f))\} \\ in_T & = & \{((v,f),f) \mid (v,f) \in N_T\} \end{array}$$

and $part(S) = T$ is defined by:

$$\begin{array}{rcl} N_T & = & N_S/{\sim_S} \\ E_T & = & \{([v]_{\sim_S}, a, [w]_{\sim_S}) \mid (v,a,w) \in E_S\} \\ nd_T & = & \{([v]_{\sim_S}, (\sum_{v\sim_S w} nd_S(w))/\underline{\mathbf{M}}) \mid v \in N_S\} \\ in_T & = & \{([v]_{\sim_S}, in_S(v)) \mid v \in N_S\} \end{array}$$

$T \lhd S$ means that T is essentially obtained from S by assigning normalised incoming edge multiplicities and positive normalised node multiplicities to the nodes of S. This may result in S-nodes disappearing (if they otherwise would have multiplicity ${}^{=}0$) or being split (if there is a choice of incoming edge multiplicities). The conditions on T ensure that it satisfies Clauses 2 and 3 of Def. 18. $part(S)$, on the other hand, combines $\sim_S$-similar nodes, and so ensures Clause 4 of the definition provided that S already satisfies Clauses 1–3.

An example can be found in Fig. 6, which shows the normalisation of the shape obtained by transforming S using the materialisation in Fig. 4. This normalisation contains four shapes, two of which (on the right hand side) contain a sub-structure consisting of one or more n-linked C-nodes disconnected from the rest of the buffer. Such a structure does not model any graph occurring on the concrete level; it is an example of the ambiguity introduced by abstraction.

The canonical shape of an arbitrary deterministic graph is defined through a mapping $can\colon \mathbf{DGra}_L \to \mathbf{CSha}_L$, defined by

$$can : G \mapsto part(S_G^{inst}) \tag{2}$$

where $S_G^{inst} = (N_G, E_G, nd, in)$ is the "instance shape" of G, defined such that nd assigns ${}^{=}1$ to all nodes $v \in N$ and $in(v)(a) = \mu$ is the unique multiplicity in $\underline{\mathbf{M}}$ such that $(tgt_G^{-1}(v) \cap lab_G^{-1}(a)) : \mu$. For instance, the shape in Fig. 1 is the image under can of the graph in the same figure. The following results are recalled from [16].

Theorem 19 *For arbitrary $G \in \mathbf{DGra}_L$, $can(G) \in \mathbf{CSha}_L$ and $\exists s\colon G \to can(G)$.*

Theorem 20 *For arbitrary $S \in \mathbf{Sha}_L$, $norm(S) = \{can(G) \mid \exists s\colon G \to S\}$.*

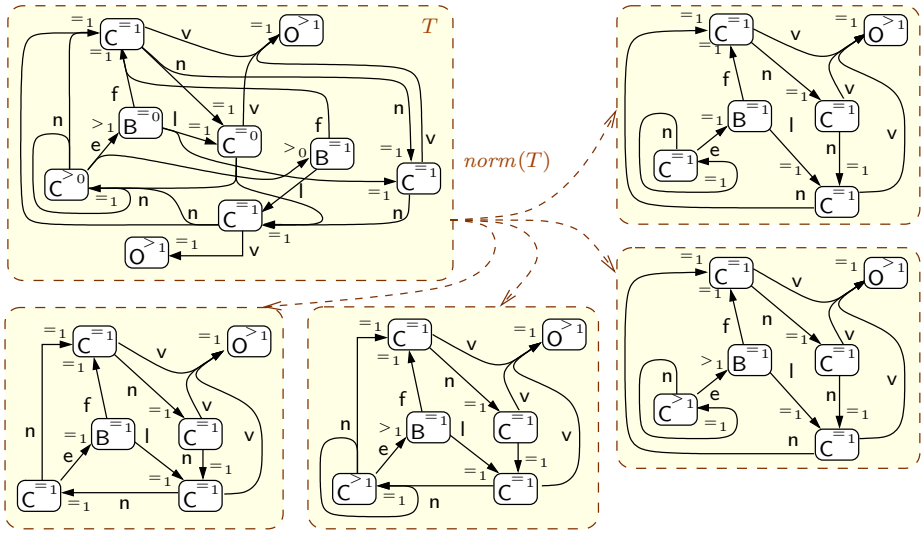

Figure 6: Normalisation of the shape T with $S^{+p} \xrightarrow{\langle \mathsf{put} \rangle, id_L} T$ (with S and p as in Fig. 4)

6 Transitions

In this section we combine the definitions and results of the previous, by defining concrete (graph) transition systems and abstract (shape) transition systems and proving their correspondence.

Definition 21 (transition system) *Let Π be a set of production rules.*

- *A* graph transition *is a triple $G \xrightarrow{P} H$ with $G, H \in \mathbf{DGra}_L$ and $P \in \Pi$ such that $G \xrightarrow{P,m} H$ for some m. A* graph transition system *is a tuple $(\mathbf{G}, \rightarrow)$ where $\rightarrow$ is the graph transition relation and $\mathbf{G} \subseteq \mathbf{DGra}_L$ is closed under $\rightarrow$ (i.e., $G \in \mathbf{G}$ and $G \xrightarrow{P} H$ implies $H \in \mathbf{G}$).*
- *A* shape transition *is a triple $S \xrightarrow{P} T$ with $S, T \in \mathbf{CSha}_L$ and $P = (L, R) \in \Pi$ such that S^{+p} is consistent, $S^{+p} \xrightarrow{P, id_L} S'$ and $T \in can(S')$ for some pre-shaping $p\colon L \rightarrow S$. A* shape transition system *is a tuple $(\mathbf{S}, \rightarrow)$ where $\rightarrow$ is the shape transition relation and $\mathbf{S} \subseteq \mathbf{CSha}_L$ is closed under $\rightarrow$.*

Given a set of production rules Π and a graph $G \in \mathbf{DGra}$, we write $GTS(\Pi, G)$ for the smallest graph transition system including G; likewise, given $S \in \mathbf{CSha}_L$ we write $STS(\Pi, S)$ for the smallest shape transition system including S. For instance, Fig. 2 shows the graph transition system $GTS(\Pi, G)$ where $\Pi = \{\langle \mathsf{put} \rangle, \langle \mathsf{get} \rangle\}$ and G is the graph of Fig. 1. Fig. 7 shows $STS(\Pi, can(G))$, where we have used some notational conventions to represent multiplicities: thin arrows and nodes are singular (node/incoming edge multiplicity $^{=}1$) whereas fat ones are multiple (multiplicity $^{>}1$). The arrows in Fig. 7 indicate $\langle \mathsf{put} \rangle$-applications; for each arrow there is an implicit $\langle \mathsf{get} \rangle$-application in the reverse direction. The darker (shaded) area is the fragment of the state space that actually is the image of the concrete transition system.

The following theorem states that can maps each graph transition system homomorphically to a finite shape transition system, and that, with respect to this mapping, $STS(\Pi, G)$ contains no spurious transitions.

Theorem 22 *Let Π be a set of production rules and $I \in \mathbf{DGra}$; let $GTS(\Pi, I) = (\mathbf{G}, \rightarrow)$ and $STS(\Pi, can(I)) = (\mathbf{S}, \rightarrow)$.*

1. *$can(\mathbf{G}) \subseteq \mathbf{S}$ and $\mathbf{S}$ is finite;*

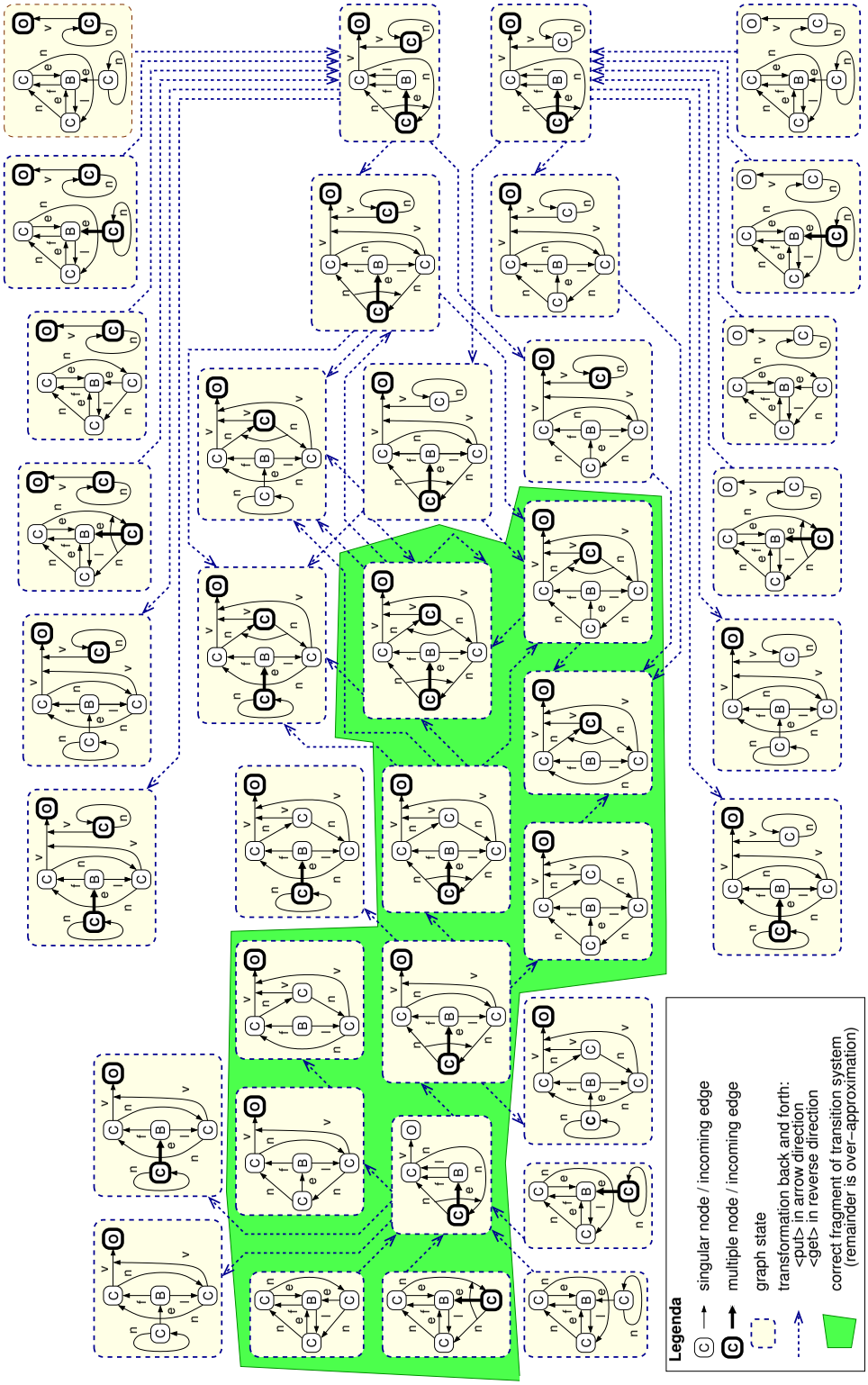

Figure 7: Abstract buffer transition system

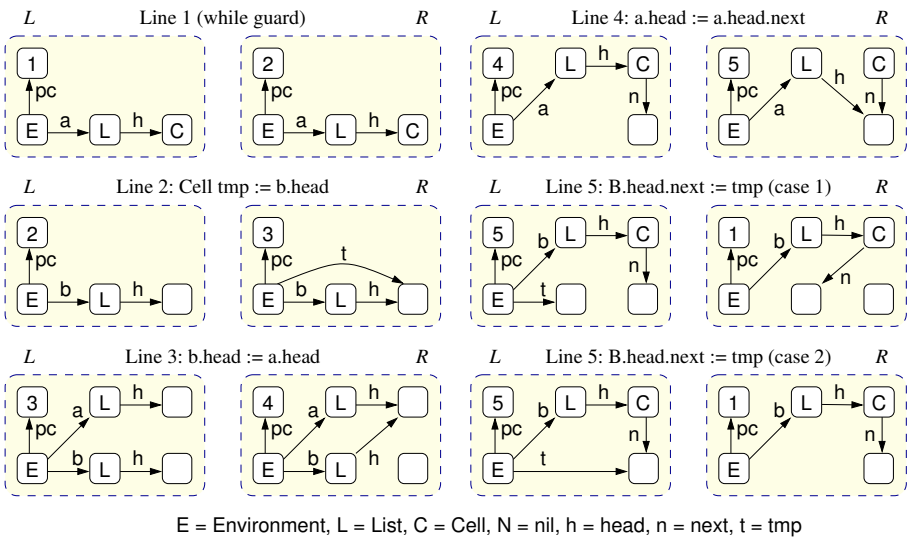

Figure 8: Small-step transformation rules for the list reversal program.

2. *For all* $G, H \in \mathbf{G}$, $G \xrightarrow{P} H$ *implies* $can(G) \xrightarrow{P} can(H)$.
3. *For all* $S, T \in \mathbf{S}$ *such that* $S \xrightarrow{P} T$, *there are* $G', H' \in \mathbf{DGra}$ *such that* $S = can(G')$ *and* $G' \xrightarrow{P} H'$.

This theorem implies that we can verify LTL safety properties, where the propositions are graph predicates in the fragment of first-order logic that is reflected by our abstraction — characterised in [14] as a fragment of 2-variable logic. Typical examples of such properties are *state invariants*, such as:

- The buffer is either empty (i.e., no cell reachable from the first contains an object), or the first cell contains an object;
- If the buffer is empty, then the last cell is the predecessor of the first;
- If a cell contains an object, then either it is the last or the next also contains an object.

Examples of valid properties that can *not* be verified, i.e., that appear to be violated on the abstract level but are in fact true in any concrete instance (often called "false negatives") are:

- All cells of the circular buffer are connected;
- ⟨put⟩ can only be executed infinitely often if ⟨get⟩ is also done infinitely often;
- Objects are removed in the order they were inserted.

List reversal. To enable a better comparison with existing approaches, the remainder of this section is devoted to an example that has been used several times before in heap structure analysis; see, e.g., [19, 18]. The program uses a data structure consisting of List-nodes pointing via a head-edge to a list of Cell-nodes linked by next-edges; there is a unique nil-node modelling the end of the list.

```
1 while (a.head != nil ) do
2     Cell tmp := b.head;
3     b.head := a.head;
4     a.head := a.head.next;
5     b.head.next := tmp;
6 od
```

Fig. 8 shows a straightforward, line-by-line translation of this program into graph transformation rules. The variables and fields are represented by edges and their values by nodes. There is a central, E-labelled node that stands for the run-time environment, to which the local variable edges are attached and which maintains a pc-labelled "program counter" edge. Line 5 needs two rules, to distinguish the case where b.head.next is already

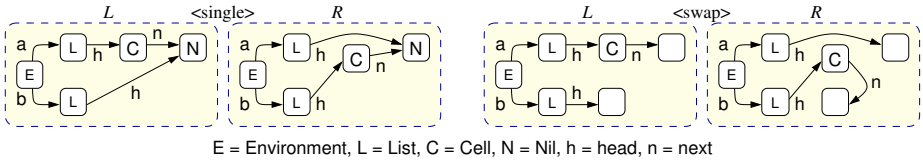

Figure 9: Large-step rules for the list reversal program.

equal to tmp (which may occur if the list a originally has only a single element); this is because our matchings are required to be injective (see Def. 2). We can now use standard graph transformation theory to concatenate these rules into larger ones, which describe the combined effect of the loop body. This results in two "large-step" rules, shown in Fig. 9 (where we have left out the program counter, which now always stands at 1).

We show in Fig. 10 the complete transition system generated by the large-step rule ⟨swap⟩ — ⟨single⟩ is never enabled from the chosen start state. Note that the transition system is smaller than the one we would get from the small-step rules (see, e.g., [19]): the graph transformation theory has paid off here. The possible runs of the transition system all terminate in S_7, which represents the reversed list which is now pointed to by b, whereas a is empty. An example property that can be seen in the transition system that the two lists are always kept separate: no Cell node is ever shared.

7 Conclusions

We have presented a technique for the push-button construction of a finite abstract model of operational semantics, on the basis of a graph production system consisting of a set of graph transformation rules. As pointed out in the introduction, the contribution with respect to previous work is that this paper works out the transformation itself and the ensuing abstract transition system (Sections 4 and 6): the shape model was presented before. Given the fact that, as argued elsewhere (see, e.g., [2, 6, 9, 11]), graph transformations are a very suitable formalism to model the behaviour of software systems, especially in the face of dynamic evolution, the results of this paper form an important step in creating a practically feasible method for the verification of such systems.

Related work. In addition to the more or less related work mentioned above, there are some lines of research that should be described in some more detail.

First and foremost among these is the work on shape graphs in [19, 20]. Although we have carried out our investigation in the context of a different formalism, there are clear

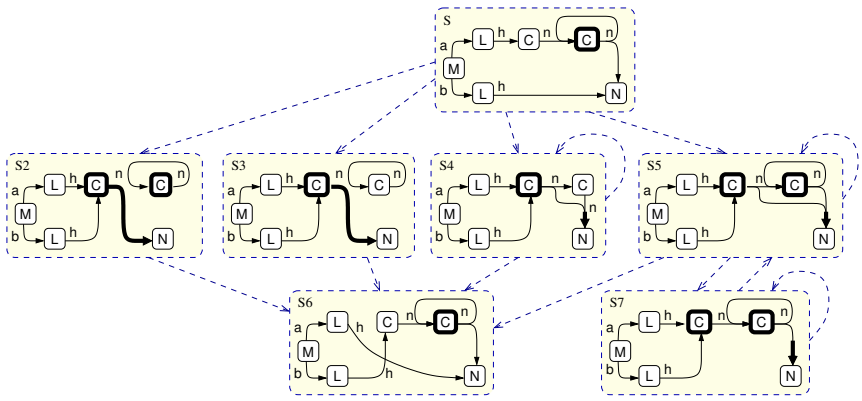

Figure 10: Abstract transition system of the list reversal program.

analogies between the shapes as presented here and those in the papers above. Technically, the main difference lies in our use of multiplicities, based on [14], rather than three-valued logic. This lends itself to another kind of abstraction refinement, different from instrumentation: extending the set of base multiplicities $\underline{\mathbf{M}}$, for instance to $\{^{=}0, ^{=}1, ^{=}2, ^{>}2\}$, does not affect the theory and will improve the precision of the abstraction. Methodologically, the difference is larger, and this is where our main contribution lies: we are using a "pure" graph transformation approach, which allows us to benefit from existing theory from that area. One place where this is apparent in the current paper is in constructing the large-step rules in Fig. 9 from the small-step rules in Fig. 8.

It should also be remarked that there are properties that we can *not* conclude from our encoding of the list example that other approaches do treat, such as the fact that no nodes become disconnected as a result of the list reversal (although it follows from the multiplicities that any such disconnected cells must be on a cycle). The reason is essentially that our abstraction reflects only a fragment of first-order logic, and hence connectivity properties cannot be verified.

More broadly speaking, our approach can be seen as an instance of abstract interpretation, pioneered by Cousot and Cousot [4]; see also [5] for a discussion of the use of abstract interpretation in model checking. In terms of [13], our shapes form a distinctness domain; however, in that terminology our abstract domain consists not of individual shapes but of *sets* of shapes (modulo isomorphism), and the abstract ordering is set inclusion. We therefore do have a Galois connection; but then, since we are not interested in computing fixpoints of computations but rather in expressing temporal properties of behaviour, we do not currently derive much benefit from this fact.

Another related area is the assertional approach for local reasoning on memory structures developed in, e.g., separation logic [12, 18]. Here, too, an abstraction of a graph-based memory representation is taken as the basic model upon which verification is carried out. Although the core formalism is quite different in this case, one possible way to combine strengths is to investigate assertional semantics for graph transformation rules.

In the context of graph transformation, the closest related work is [1] on approximation of graph transition systems using *unfolding*, a technique that is generalised from Petri nets. Instead of constructing individual states, an unfolding combines all states into a single structure, in which transitions are modelled as purely local changes. Since eventually such local changes tend to propagate to a global level, the unfolding is *cut off* after a certain number of steps, at which point an over-approximation of the remaining behaviour is taken. Essentially, this approach promises the same capabilities for generic and infinite-state system verification as ours; once tool support for both is in place, a more detailed comparison should prove very interesting.

Future work. There is a host of smaller and larger improvements to be made.

- The current framework has a number of limitations in the graphs and transformation rules that are supported: graphs are deterministic, matchings have a dangling edge condition and have to be injective, and negative application conditions (cf. [10]) are not allowed. We conjecture that all of these restrictions can be lifted to some degree, at the price of some complications in the theory. For instance, rather than forbidding transformations that would violate the determinism, as we currently do in the definition of concrete matchings, one could take the pushout in the category of deterministic graphs, which essentially means determinising the graph after transformation, i.e., recursively merging outgoing edges with the same label.

- Graph transformations enjoy a very strong algebraic theory (see, e.g., [3]), which we have completely ignored in the current paper. In particular, our abstract shape transformation have no underlying notion of a morphism or span of morphisms; instead they are based on *ad hoc* constructions. Consequently, there is no way to lift the

results of this paper to other graph formalisms (for instance typed, attributed, or hypergraphs) or other types of abstraction without redoing the proofs. Working out an algebraic theory of abstract graph transformations is one of the items on our agenda.

- Since (as a consequence of the previous point) shape transitions do not include a relation between the nodes of source and target shapes, we cannot keep track of the identities of nodes. Hence certain types of properties cannot be verified that we did study, for more limited pointer structures, in [7], such as for instance the existence of a permanent link between two particular nodes. Here, too, we are quite interested in regaining the lost ground.

Notwithstanding the fact that there is ample room for improvement, the constructions worked out in this paper are mature enough for implementation. We plan to extend the tool GROOVE (see [15]), which has the capability of generating concrete state spaces from graph production systems for the purpose of model checking (see [17]), with the necessary functionality to deal with shapes. As a proof-of-concept, we have "hand-crafted" the examples presented in this paper into GROOVE production rules mimicking the abstract behaviour.

References

[1] P. Baldan, B. König, and B. König. A logic for analyzing abstractions of graph transformation systems. In R. Cousot, editor, *Static Analysis*, volume 2694 of *Lecture Notes in Computer Science*, pages 255–272. Springer-Verlag, 2003.

[2] A. Corradini, F. L. Dotti, L. Foss, and L. Ribeiro. Translating java into graph transformation systems. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *Second International Conference on Graph Transformation*, volume 3256 of *Lecture Notes in Computer Science*, pages 383–389. Springer-Verlag, 2004.

[3] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation, part I: Basic concepts and double pushout approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume I: Foundations, chapter 3, pages 163–246. World Scientific, Singapore, 1997.

[4] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY.

[5] P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Automated Software Engineering*, 6(1):69–95, 1999.

[6] R. Depke, R. Heckel, and J. M. Küster. Formal agent-oriented modeling with UML and graph transformation. *Science of Computer Programming*, 44:229–252, 2002.

[7] D. Distefano, J.-P. Katoen, and A. Rensink. Who is pointing when to whom? On the automated verification of linked list structures. In *The 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, Lecture Notes in Computer Science. Springer-Verlag, 2004. To appear.

[8] D. Distefano, A. Rensink, and J.-P. Katoen. Model checking birth and death. In R. A. Baeza-Yates, U. Montanari, and N. Santoro, editors, *Foundations of Information Technology in the Era of Network and Mobile Computing*, volume 223 of *IFIP Conference Proceedings*, pages 435–447. Kluwer Academic Publishers, 2002.

[9] F. L. Dotti, L. Foss, L. Ribeiro, and O. M. dos Santos. Verification of distributed object-based systems. In E. Najm, U. Nestmann, and P. Stevens, editors, *Formal Methods for Open Object-based Distributed Systems*, volume 2884 of *Lecture Notes in Computer Science*, pages 261–275. Springer-Verlag, 2003.

[10] A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 26(3/4):287–313, 1996.

[11] S. Kuska, M. Gogolla, R. Kollmann, and H.-J. Kreowski. An integrated semantics for UML class, object and state diagrams based on graph transformation. In M. Butler, L. Petre, and K. Sere, editors, *IFM 2002*, volume 2235 of *Lecture Notes in Computer Science*, pages 11–28. Springer-Verlag, 2002.

[12] P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In L. Fribourg, editor, *CSL 2001*, volume 2142 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, 2001.

[13] I. Pollet, B. L. Charlier, and A. Cortesi. Distinctness and sharing domains for static analysis of java programs. In J. L. Knudsen, editor, *ECOOP 2001 - Object-Oriented Programming, 15th European Conference, Budapest, Hungary, June 18-22, 2001, Proceedings*, volume 2072 of *Lecture Notes in Computer Science*, pages 77–98. Springer-Verlag, 2001.

[14] A. Rensink. Canonical graph shapes. In D. A. Schmidt, editor, *Programming Languages and Systems — European Symposium on Programming (ESOP)*, volume 2986 of *Lecture Notes in Computer Science*, pages 401–415. Springer-Verlag, 2004.

[15] A. Rensink. The GROOVE simulator: A tool for state space generation. In J. Pfalz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, volume 3063 of *Lecture Notes in Computer Science*, pages 479–485. Springer-Verlag, 2004.

[16] A. Rensink. State space abstraction using shape graphs. In *Automatic Verification of Infinite-State Systems (AVIS)*, Electronic Notes in Theoretical Computer Science. Elsevier, 2004. To appear.

[17] A. Rensink, Á. Schmidt, and D. Varró. Model checking graph transformations: A comparison of two approaches. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *International Conference on Graph Transformations (ICGT)*, volume 3256 of *Lecture Notes in Computer Science*, pages 226–241. Springer-Verlag, 2004.

[18] J. Reynolds. Separation logic: A logic for shared mutable data structures. In *Seventeenth Annual IEEE Symposium on Logic in Computer Science*. IEEE, Computer Society Press, 2002.

[19] M. Sagiv, T. Reps, and R. Wilhelm. Solving shape-analysis problems in languages with destructive updating. *ACM Trans. Prog. Lang. Syst.*, 20(1):1–50, Jan. 1998.

[20] M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Prog. Lang. Syst.*, 24(3):217–298, May 2002.

A Proofs of the theorems

Proposition 11 *Let $L, G \in \mathbf{DGra}_L$ and $S \in \mathbf{Sha}_L$. If $f\colon L \to G$ is an injective morphism and $s\colon G \to S$ a shaping, then $s \circ f$ is a pre-shaping of L into S.*

Proof. Let $p = s \circ f$. Clearly p is a graph morphism; we only have to show satisfaction of the multiplicities' upper bounds, in the sense of Def. 10.

- Let $v \in N_S$ be arbitrary. It follows (by the fact that s is a shaping) that $s^{-1}(v) : nd_S(v)$, which implies (among other things) $|s^{-1}(v)| \leq \lceil nd_S(v) \rceil$. From the injectivity of f it follows that $|f^{-1}(s^{-1}(v))| \leq |s^{-1}(v)|$, hence we are done.
- Let $v \in N_L$ and $a \in L$ be arbitrary. Since s is a shaping it follows (among other things) that $|\{w \in N_G \mid (w, a, f(v)) \in E_G\}| \leq \lceil in_S(s(v))(a) \rceil$. From the injectivity of f it follows that

  $$|\{w \in N_L \mid (f(w), a, f(v)) \in E_G\}| \leq |\{w \in N_G \mid (w, a, f(v)) \in E_G\}| \ ;$$

  moreover, since f is a graph morphism we have $\{w \in N_L \mid (w, a, v) \in E_L\} \subseteq \{w \in N_L \mid (f(w), a, f(v)) \in E_G\}$. These three inequalities suffice to conclude the proof obligation. □

Proposition 12 *Let $L \in \mathbf{DGra}_L$ and $S \in \mathbf{Sha}_L$ and let $c\colon L \to S$ be a concrete pre-shaping. For any $G \in \mathbf{DGra}_L$ with a shaping $s\colon G \to S$, there is an injective morphism $m\colon L \to G$ such that $c = s \circ m$.*

Proof. By the fact that c is concrete, it follows that $nd_S(c(v)) = {}^{=}1$ for all $v \in N_L$; hence $s(w) = c(v)$ uniquely identifies $w \in N_G$, and so $s^{-1}(c(v))$ is well-defined. We make use of this fact by defining a node mapping $m\colon N_L \to N_G$ as

$$m : v \mapsto s^{-1}(c(v)) \ .$$

Since $c = s \circ m$ holds by construction, we only have to show that m is an injective graph morphism.

Let $(v, a, w) \in E_L$ be arbitrary. It follows that $(c(v), a, c(w)) \in E_S$; but then (due to Clause 3 of Def. 7) $\exists (m(v), a, w') \in N_G$, and so $(c(v), a, s(w')) \in E_S$. Because c is concrete, it follows that $s(w') = c(w)$, and so $w' = m(w)$. We may conclude that m is indeed a graph morphism.

The injectivity of m follows from the fact that c is injective (which is enforced by the node multiplicity $nd_S(c(v)) = {}^{=}1$ for all $v \in N_L$). □

Proposition 13 *Let $L \in \mathbf{DGra}_L$ and $S \in \mathbf{Sha}_L$, and let $p\colon L \to S$ be a pre-shaping. α_p gives rise to an abstraction morphism from S^{+p} to S, and id_L gives rise to a concrete pre-shaping of L into S^{+p}, such that $p = \alpha_p \circ id_L$.*

Proof.α_p is a graph morphism by construction of E^{+p}. To show that $\alpha_p\colon S^{+p} \to S$ is an abstraction morphism we prove the properties of Def. 8.

1. Let $v \in N_S$; then $\alpha_p^{-1}(v) = p^{-1}(v) \cup \{v\}$. Defining $i = |p^{-1}(v)|$ and using Prop. 5.2 we obtain

   $$\sum nd^{+p}(\alpha_p^{-1}(v)) = (nd_S(v) - i) + {}^{=}i \subseteq nd_S(v) \ .$$

2. By construction of in^{+p};
3. By construction of E^{+p}.

id_L is a concrete pre-shaping by construction of S^{+p}, taking into account that p is already a shaping. Finally, $p = \alpha_p \circ id_L$ is immediate by the definition of α_p. □

Proposition 14 *Let $L, G \in \mathbf{DGra}_L$ and $S \in \mathbf{Sha}_L$. For an arbitrary injective morphism $m\colon L \to G$ and a shaping $s\colon G \to S$, let $p = s \circ m$; then there is a shaping $t\colon G \to S^{+p}$ with $s = \alpha_p \circ t$ and $t \circ m = id_L$.*

Proof. Note that p is a pre-shaping by Prop. 11, so the materialisation S^{+p} is well-defined. The required shaping t is given by

$$t : v \mapsto \begin{cases} m^{-1}(v) & \text{if } v \in m(N_L) \\ s(v) & \text{otherwise.} \end{cases}$$

On the level of functions over node sets, we show $s = \alpha_p \circ t$ by a simple case distinction. Let $v \in N_G$ be arbitrary.

- If $v \in m(N_L)$ then $t(v) = m^{-1}(v) \in N_L$, implying $\alpha_p(t(v)) = p(t(v)) = s(m(m^{-1}(v))) = s(v)$.
- If $v \notin m(N_L)$ then $t(v) = s(v)$, implying $\alpha_p(t(v)) = id_S(t(v)) = s(v)$.

To see that $t \circ m = id_L$ holds (as functions over node sets), let $v \in N_L$ be arbitrary; then $m(v) \in m(N_L)$, hence $t(m(v)) = m^{-1}(m(v)) = v$. We now show that t is a shaping of G to S^{+p}.

- By construction, t maps N_G into N^{+p}.
- Let $(v, a, w) \in E_G$ be arbitrary. It follows that $(s(v), a, s(w)) \in E_S$ due to the fact that s is a shaping; hence $(t(v), a, t(w)) \in \alpha_p^{-1}(E_S)$ due to $s = \alpha_p \circ t$, proved above. To show that $(t(v), a, t(w)) \in E^{+p}$, we now only have to show that either $t(v) \notin N_L$ or $\nexists (t(v), a, w') \in E_L : w' \neq t(w)$. For this purpose assume $t(v) \in N_L$ and $(t(v), a, w') \in E_L$; then $(m(t(v)), a, m(w')) \in E_G$. It follows by construction of t that $v \in m(N_L)$ and $v = m(t(v))$; hence $m(w') = w$ due to the determinism of G, implying $t(m(w')) = t(w)$. Since $t(m(w')) = w'$ by construction of t, we are done.
- Let $v \in N^{+p}$ be arbitrary. We make the following case distinction:
  - $v \in N_L$. By definition, $t^{-1}(v) = \{m(v)\}$; since $nd^{+p}(v) = {}^{=}1$, we are done.
  - $v \in N_S$. By definition, $t^{-1}(v) = s^{-1}(v) \setminus m(N_L)$. Since m is injective, $|s^{-1}(v) \cap m(N_L)| = |m^{-1}(s^{-1}(v))| = |p^{-1}(v)|$. Since $s^{-1}(v) : nd_S(v)$ by the fact that s is a shaping, it follows by Prop. 5.1 that $t^{-1}(v) : nd^{+p}(v)$.
- Let $v \in N_G$ and $a \in L$ be arbitrary, and define the a-predecessors of v in G as $X = \{w \in N_G \mid (w, a, v) \in E_G\}$. By the fact that s is a shaping it follows that $X : in(s(v))(a)$. Due to $s = \alpha_p \circ t(v)$ it follows that $in^{+p}(t(v)) = in^{+p}(\alpha_p(t(v))) = in^{+p}(s(v))$, hence $X : in^{+p}(t(v))(a)$. □

Fig. 11 shows a diagram to clarify Theorems 16 and 17.

Theorem 16 *Let $P = (L, R) \in \mathbf{Prod}_L$ and $S \in \mathbf{Sha}_L$, and assume $S \xrightarrow{P,c} T$. For any shaping $s\colon G \to S$, there exists a matching m for P in G such that $c = s \circ m$, and $G \xrightarrow{P,m} H$ such that there is a shaping $t\colon H \to T$.*

Proof. Let $s\colon G \to S$ be an arbitrary shaping. By $S \xrightarrow{P,c} T$ we have that $c\colon L \to G_S$ is a concrete pre-shaping. Then, by Prop. 12 there exists an injective morphism $m\colon L \to G$ such that $c = s \circ m$. We show that m is in fact a (concrete) matching. To show this, we prove that m satisfies the conditions of Def. 3.

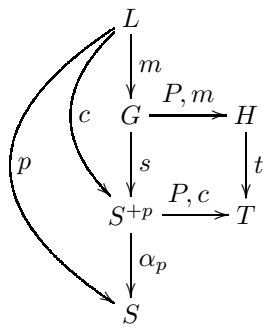

Figure 11: Concrete and abstract transitions; visualization of Theorems 16 and 17.

1. Let $(v, a, w) \in E_G$; this implies $(s(v), a, s(w)) \in E_S$. Since c is an abstract matching, we have:

$$s(v) \in c(N^{\mathsf{del}}) \vee s(w) \in c(N^{\mathsf{del}}) \;\Rightarrow\; (s(v), a, s(w)) \in c(E^{\mathsf{del}})$$
implying
$$c^{-1}(s(v)) \in N^{\mathsf{del}} \vee c^{-1}(s(w)) \in N^{\mathsf{del}} \;\Rightarrow\; c^{-1}((s(v), a, s(w))) \in E^{\mathsf{del}}$$
implying [since $c^{-1} \circ s = m^{-1}$]
$$m^{-1}(v) \in N^{\mathsf{del}} \vee m^{-1}(w) \in N^{\mathsf{del}} \;\Rightarrow\; m^{-1}((v, a, w)) \in E^{\mathsf{del}}$$
implying
$$v \in m(N^{\mathsf{del}}) \vee w \in m(N^{\mathsf{del}}) \;\Rightarrow\; (v, a, w) \in m(E^{\mathsf{del}}).$$

2. Again, if $(v, a, w) \in E_G$, then $(s(v), a, s(w)) \in E_S$. Since c is an abstract matching, we have:

$$s(v) \in c(N^{\mathsf{use}}) \wedge \exists(c^{-1}(s(v)), a, w') \in E^{\mathsf{new}} \;\Rightarrow\; (s(v), a, s(w)) \in c(E^{\mathsf{del}})$$
implying
$$c^{-1}(s(v)) \in N^{\mathsf{use}} \wedge \exists(c^{-1}(s(v)), a, w') \in E^{\mathsf{new}} \;\Rightarrow\; c^{-1}((s(v), a, s(w))) \in E^{\mathsf{del}}$$
implying [since $c^{-1} = m^{-1} \circ s^{-1}$]
$$m^{-1}(v) \in N^{\mathsf{use}} \wedge \exists(m^{-1}(v), a, w') \in E^{\mathsf{new}} \;\Rightarrow\; m^{-1}(v, a, w) \in E^{\mathsf{del}}$$
implying
$$v \in m(N^{\mathsf{use}}) \wedge \exists(m^{-1}(v), a, w') \in E^{\mathsf{new}} \;\Rightarrow\; (v, a, w) \in m(E^{\mathsf{del}})$$

This proves that m is a (concrete) matching. Hence, there exists a transition $G \xrightarrow{P,m} H$ where by Def. 3 we have:

$$\begin{aligned} N_H &= (N_G \setminus m(N^{\mathsf{del}})) \cup N^{\mathsf{new}} \\ E_H &= (E_G \setminus m(E^{\mathsf{del}})) \cup E^{\mathsf{new}} \end{aligned}$$

where N^{new} and E^{new} are fresh by the assumption on the definition of production rule.

It remains to prove that there exists a shaping t: $H \to T$. This is defined by the following node function:

$$t : v \mapsto \begin{cases} v & \text{if } v \in N^{\mathsf{new}} \\ s(v) & \text{otherwise.} \end{cases}$$

We prove that t is indeed a shaping.

- t is a graph morphism. To see this, let $(v, a, w) \in E_H$. If $(v, a, w) \in E_G \setminus m(E^{\mathsf{del}})$ we have: $t(v, a, w) = s(v, a, w) = (s(v), a, s(w)) = (t(v), a, t(w))$ since s is a

shape morphism. Otherwise if $(v, a, w) \in E^{\mathsf{new}}$ then we have to distinguish several cases depending whether v, w belong to N^{new} or to N^{use}. In all cases, it is trivial to see that by construction we have $t((v, a, w)) = (f(v), a, f(w)) = (t(v), a, t(w))$.

- Now we show that conditions 1-3 of Def. 7 hold.

  1. If $v \in N^{\mathsf{new}}$ then $|t^{-1}(v)| = 1 \in nd_T(v)$. If $v \in N_S$ then, since s is a shaping, we have $|t^{-1}(v)| = |s^{-1}(v)| \in nd_S(v) = nd_T(v)$.
  2. Let $v \in N_H$ and $a \in L$. If $v \in N^{\mathsf{new}}$ then $in_T(t(v))(a) = in_T(v)(a)$, which equals $^{=}|\{w \mid (w, a, v) \in E^{\mathsf{new}}\}|$. However, since $v \in N^{\mathsf{new}}$ we have $\{w \mid (w, a, v) \in E^{\mathsf{new}}\} = \{w \mid (w, a, v) \in E_H\}$. $\subseteq$ is trivial. We show $\supseteq$ by contradiction. Assume $\exists (w, a, v) \in E_H \setminus E^{\mathsf{new}}$ then $(w, a, v) \in E_G \setminus m(E^{\mathsf{del}})$ and therefore $v \notin N^{\mathsf{new}}$ which is indeed a contradiction. Hence, we conclude that $\{w \mid (w, a, v) \in E_H\} : in_T(t(v))(a)$.
     If $v \in N_G \setminus m(N^{\mathsf{del}})$ then
     $$\begin{aligned} in_T(t(v))(a) &= in_S(s(v))(a) - |\{w \mid (w, a, s(v)) \in c(E^{\mathsf{del}})\}| \\ &\quad + {}^{=}|\{w \mid (w, a, s(v)) \in E^{\mathsf{new}}\}| \ . \end{aligned}$$
     Due to the fact that c and m are injective morphisms, and moreover $m^{-1} = c^{-1} \circ s$, we have
     $$\begin{aligned} &|\{w \in N_H \mid (w, a, v) \in E_H\}| \\ &= |\{w \in N_G \mid (w, a, v) \in E_G\}| - |\{w \in N_G \mid (w, a, v) \in m(E^{\mathsf{del}})\}| \\ &\quad + |\{w \in N_G \mid (w, a, v) \in E^{\mathsf{new}}\}| \\ &= |\{w \in N_G \mid (w, a, v) \in E_G\}| - |\{w \in N_T \mid (w, a, s(v)) \in c(E^{\mathsf{del}})\}| \\ &\quad + |\{w \in N_T \mid (w, a, s(v)) \in E^{\mathsf{new}}\}| \ . \end{aligned}$$
     Since s is a shaping, we have $|\{w \in N_G \mid (w, a, v) \in E_G\}| \in in_S(s(v))(a)$; hence we may conclude $|\{w \in N_H \mid (w, a, v) \in E_H\}| \in in_T(t(v))(a)$.
  3. Let $v \in N_H,\ a \in L$ and $(t(v), a, w) \in E_T$. If $(t(v), a, w) \in E^{\mathsf{new}}$ then by construction $(t(v), a, w) \in E_H$.
     If $(t(v), a, w) \in E_S \setminus c(E^{\mathsf{del}})$ then $t(v) \in N_S \setminus c(N^{\mathsf{del}})$. By definition of t it follows that $v \notin N^{\mathsf{new}}$ which implies $t(v) = s(v)$. Therefore we have $(t(v), a, w) = (s(v), a, w)$. Since s is a shaping by hypothesis, then there exists $(v, a, w') \in E_G$ for some w' such that $s(w') = w$. Thus to show that $(v, a, w') \in E_H$ it remains to be proved that $(v, a, w') \notin m(E^{\mathsf{del}})$. We prove that by contradiction. Assume $(v, a, w') \in m(E^{\mathsf{del}})$ then $m^{-1}((v, a, w')) \in E^{\mathsf{del}}$. Since $m^{-1} = c^{-1} \circ s$ then it follows $s(v, a, w') \in c(E^{\mathsf{del}})$ which implies $(s(v), a, s(w')) \in c(E^{\mathsf{del}})$. This finally implies $(t(v), a, w) \in c(E^{\mathsf{del}})$ that contradicts our initial assumption.

  Hence, we conclude that t is a shaping. □

Theorem 17 *Let $P = (L, R) \in \mathbf{Prod}_L$ and $G \in \mathbf{DGra}_L$, and assume $G \xrightarrow{P,m} H$. For any shaping $s\colon G \to S$ such that $c = s \circ m$ is concrete, $S \xrightarrow{P,c} T$ such that there is a shaping $t\colon H \to T$.*

Proof. Since there exists $m\colon L \to G$ and $s\colon G \to S$ then by Prop. 11 $s \circ m\colon L \to S$ is a pre-shaping. Let $c = s \circ m$ be concrete. Because m is a concrete matching, we can prove that c is an abstract matching for P in the graph part of S. To show this we prove condition 1 and 2 of Def. 3. Let $(v, a, w) \in E_S$, we have $(c^{-1}(v), a, c^{-1}(w)) \in E_L$. Since $c^{-1} = m^{-1} \circ s^{-1}$ we have:

$$(s^{-1}(v), a, s^{-1}(w)) \in E_G.$$

Since m is a concrete matching we have:

1. For the first condition

$$s^{-1}(v) \in m(E^{\mathsf{del}}) \vee s^{-1}(w) \in m(E^{\mathsf{del}}) \;\Rightarrow\; (s^{-1}(v), a, s^{-1}(w)) \in m(E^{\mathsf{del}})$$

implying

$$m^{-1}(s^{-1}(v)) \in E^{\mathsf{del}} \vee m^{-1}(s^{-1}(w)) \in E^{\mathsf{del}} \\ \Rightarrow\; (m^{-1}(s^{-1}(v)), a, m^{-1}(s^{-1}(w))) \in E^{\mathsf{del}}$$

implying [since $c^{-1} = m^{-1} \circ s^{-1}$]

$$c^{-1}(v) \in E^{\mathsf{del}} \vee c^{-1}(w) \in E^{\mathsf{del}} \;\Rightarrow\; (c^{-1}(v), a, c^{-1}(w)) \in E^{\mathsf{del}}$$

implying

$$v \in c(E^{\mathsf{del}}) \vee w \in c(E^{\mathsf{del}}) \;\Rightarrow\; (v, a, w) \in c(E^{\mathsf{del}})$$

2. For the second condition:

$$s^{-1}(v) \in m(N^{\mathsf{use}}) \wedge \exists (m^{-1}(s^{-1}(s)), a, w') \in E^{\mathsf{new}} \\ \Rightarrow\; (s^{-1}(v), a, s^{-1}(w)) \in m(E^{\mathsf{del}})$$

implying

$$v \in s \circ m(N^{\mathsf{use}}) \wedge \exists (c^{-1}(s), a, w') \in E^{\mathsf{new}} \;\Rightarrow\; (v, a, w) \in s \circ m(E^{\mathsf{del}})$$

implying

$$v \in c(N^{\mathsf{use}}) \wedge \exists (c^{-1}(s), a, w') \in E^{\mathsf{new}} \;\Rightarrow\; (v, a, w) \in c(E^{\mathsf{del}})$$

Therefore we conclude that c is an abstract matching. Then, by Def. 15 there exists a transition $S \xrightarrow{P,c} T$. Moreover, The target graphs H and T of the concrete and the abstract transition are defined according to Def. 3 and Def. 15. Let $t\colon H \to T$ be defined as in the proof of Theorem 16; as shown in that proof, t is a shaping. □

Theorem 22 *Let Π be a set of production rules and $I \in \mathbf{DGra}$; let $GTS(\Pi, I) = (\mathbf{G}, \to)$ and $STS(\Pi, can(I)) = (\mathbf{S}, \to)$.*

1. *$can(\mathbf{G}) \subseteq \mathbf{S}$ and $\mathbf{S}$ is finite;*
2. *For all $G, H \in \mathbf{G}$, $G \xrightarrow{P} H$ implies $can(G) \xrightarrow{P} can(H)$.*
3. *For all $S, T \in \mathbf{S}$ such that $S \xrightarrow{P} T$, there are $G', H' \in \mathbf{DGra}$ with a shaping $s\colon G' \to S$, such that $G' \xrightarrow{P} H'$.*

Proof. Let $S = can(G)$.

1. $can(\mathbf{G}) \subseteq \mathbf{S}$ follows from the next item. The finiteness of $\mathbf{S}$ is a consequence of the finiteness of $\mathbf{CSha}_L$, proved in [14].

2. Let $P = (L, R)$ and assume m is the matching for $G \xrightarrow{P} H$. By construction of S, there is a shaping $s\colon G \to S$ (see Th. 19). Due to Prop. 11, $p = s \circ m$ is a pre-shaping of L in S; hence due to Prop. 13, there is a concrete pre-shaping $id_L\colon L \to S^{+p}$. Hence due to Th. 17, there is an abstract shape transformation $S^{+p} \xrightarrow{P, id_L} T$ with a shaping $t\colon H \to T$. By Th. 20 we have $can(H) \in norm(T)$. It follows that, by definition, $S \xrightarrow{P} T$.

3. Let $P = (L, R)$. By definition of shape transitions, S^{+p} is consistent, $S^{+p} \xrightarrow{P, id_L} S'$ and $T \in can(S')$ for some pre-shaping $p\colon L \to S$. It follows that there is a shaping $s'\colon G \to S^{+p}$; by Prop. 9, $s = \alpha_p \circ s'$ is then a shaping of G in S. By Th. 16, there exists a matching m for P in G such that $id_L = s \circ m$ and $G \xrightarrow{P,m} H$. □