State Space Abstraction using Shape Graphs

Arend Rensink
Department of Computer Science, University of Twente
P.O.Box 217, 7500 AE, The Netherlands
rensink@cs.utwente.nl

February 12, 2004

Abstract

To represent the individual states of software systems we propose to use edge-labelled graphs: nodes will stand for dynamically allocated entities (e.g., objects or method frames) and edges for relations between those entities (e.g., arising from associations or variables). Obviously, as these graphs may in principle grow unboundedly, the state space is generally infinite. In this paper we present a technique to automatically obtain finite approximations of arbitrary state spaces, by recording only the local structure of the individual graphs: essentially, for each node we only store the approximate number of its neighbours according to each edge label. This gives rise to a variant of *shape graphs* described elsewhere.

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 behavioural models. In this paper we describe a technique to define such approximations automatically for arbitrary *graph-based* state models, that is, models in which individual states are represented as graphs with labelled edges over a finite alphabet.

Let us first present the abstraction principle we will use. As behavioural models we take Kripke structures $T = \langle S, \iota, \pi, \rightarrow \rangle$, where S is a set of states with initial state $\iota \in S$, $\pi: S \rightarrow \mathbf{2}^{\Psi}$ (with Ψ a universe of properties) is a mapping from states to sets of properties that hold there, and $\rightarrow \subseteq S \times \times S$ is a transition relation. (The models can easily be extended with a Büchi fairness condition, with transition labels, or with a modality predicate over the transitions as in [6].) Given two transition systems T, U, we will say that U *abstracts* T if there exists a surjective mapping $\alpha: S_T \rightarrow S_U$ such that

- $\psi \in \pi_U(\alpha(s))$ implies $\psi \in \pi_T(s))$; i.e., α reflects (or weakly preserves) Ψ-properties that hold in T. This means that verifying a property on the abstract model is enough to guarantee that it holds in the concrete model.[1]
- $\alpha(\iota_T) = \iota_U$; i.e., α preserves the initial state.
- $s \rightarrow_T s'$ implies $\alpha(s) \rightarrow_U \alpha(s')$; i.e., α preserves transitions.

[1] Weak reflection can be improved by using *3-valued logic* on the abstract level, as in [12].

- $s \rightarrow_U s'$ implies $\hat{s} \rightarrow_T \hat{s}'$ for some $\hat{s} \in \alpha^{-1}(s), \hat{s}' \in \alpha^{-1}(s')$; i.e., α is surjective on transitions as well. Intuitively this means that there are no spurious transitions in the abstract model.

The last two conditions can be combined by requiring $\rightarrow_U = \{(\alpha(s), \alpha(s')) \mid s \rightarrow_T s'\}$. This implies that the effect of abstraction on transitions is determined completely by its effect on states. We therefore feel justified in concentrating on state abstraction in this paper, leaving the issue of transition abstraction to future work (see Section 4).

In this paper we study a particular class of state representations, namely where $S \subseteq \mathbf{Graph}_L$, the class of (edge-labelled) *graphs* over a fixed, finite alphabet L. We consider graphs to be particularly suitable to represent program states, since they not only naturally support the dynamic (de)allocation of objects (heap allocation) —see also Sagiv et al. [11]— but are equally suited for method frames and call chains (stack allocation). We present a general technique for graph abstraction based on collecting graph nodes with similar local structure, again à la Sagiv et al. but then for general graphs, using the concept of *canonical graph shapes* developed in [9]. Since we have shown there that the number of canonical shapes over a fixed, finite alphabet is finite, this abstraction gives rise to a finite representation of arbitrary (in particular also infinite) models.

The universe of properties Ψ we envisage for graphs is much richer than for traditional Kripke structures: we propose to use first-order logic with unary and binary predicates, which denote collections of appropriately labelled edges. The interpretation of first-order logic on shapes, however, is certainly not trivial; this, too, is outside the scope of the current paper (but see Section 4).

The paper is structured as follows. In Section 2 we introduce the basic concepts of graphs and shapes and we show how they may be used to represent program states on a concrete and abstract level. In Section 3 we define the subclass of canonical shapes; we show that every graph corresponds to a unique canonical shape, which can be constructed automatically by a node partitioning of the graph. We also give a "best effort" mapping from arbitrary shapes to canonical shapes. These two results, together with a presentation of the model of shapes that does not rely on logic, are the main contribution of this paper w.r.t. [9]. The mapping from graphs to canonical shape graphs gives rise to the abstraction α discussed above. We briefly discuss future work in Section 4.

For readability we have collected all proofs of the theorems in Appendix A.

2 Definitions

For a given set of nodes N, $N_\perp = N \cup \{\perp\}$ denotes N augmented with the "undefined value" $\perp$.

Definition 2.1 (graphs) *A* graph *is a tuple $G = \langle L, N, E\rangle$ where L is a finite set of labels, N a finite set of nodes and $E \subseteq N \times L \times N_\perp$ a set of labelled edges.*

We use $\mathbf{Graph}_L$ to denote the class of graphs G with $G_L = L$. We write $src(e), lab(e), tgt(e)$ for the source node, label, and target node of a given edge e, respectively. If $tgt(e) = \perp$ then e is actually a *unary* edge. The following defines a corresponding notion of graph morphism, which is instrumental in defining abstractions and transformations.

Definition 2.2 (morphisms) *Let $G, H \in \mathbf{Graph}_L$ be arbitrary.*

1. *A* morphism *from G to H is a strict function $\phi\colon N_G \rightarrow N_H$ such that $(\phi(p), a, \phi(q)) \in E_H$ for all $(p, a, q) \in E_G$. We write $\phi\colon G \rightarrow H$ to denote that fact that ϕ is a morphism from G to H.*
2. *G and H are* isomorphic*, denoted $G \cong H$, if there is a bijective morphism from G to H.*
3. *A* typing *of G is a morphism $\tau\colon G \rightarrow T$. We call G an* instance of *T and T a* type of *G.*

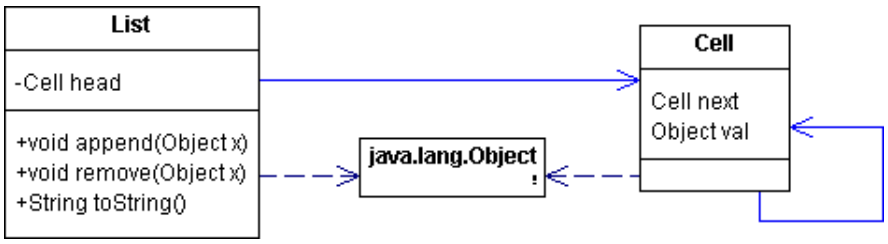

Figure 1: A list with append and remove methods

Graphs are used to model a huge variety of systems. In general, nodes stand for entities in a given domain and edges for relations between those entities. In this paper, we use graphs to model state snapshots: the entities are data objects and method frames created during the execution of a program, and the edges model object fields and local method variables. (Note, however, that the theory developed here does not depend upon that application area.) We use a tiny Java program, consisting of the two classes depicted in the class diagram of Figure 1, as a motivating example. Apart from the attributes shown in the diagram, we assume the append and remove methods have *local* variables acurr and rcurr, respectively, pointing to the "current cell" of the list.

Figure 2 shows an example graph modelling a linked list with two active methods, and another graph representing the static information of the `List` class. There is an obvious typing from the one to the other. Labels that are written inside nodes depict unary edges. In this setup, each node stands for a single object or method frame; thus, to model a given state one needs as many nodes as there are allocated objects. Clearly, this number is unbounded. A natural idea to alleviate this problem is to collect families of "similar" objects and represent them using single nodes. It follows that a node in the resulting, more abstract, graph no longer stands for a single object but (generally) for a number of them; and similar for the edges. We use *multiplicities* to record those numbers, up to a certain precision.

Definition 2.3 (multiplicities)

1. *A* multiplicity *is an interval* $(i, j) \in \mathbf{N} \times (\mathbf{N} \cup \{\omega\})$ *such that* $i \leq j$ *(where* ω *denotes infinity, hence* $n \leq \omega$ *for all* $n \in \mathbf{N}$*). The set of multiplicities is denoted* $\mathbf{M}$*. We commonly write* (i, j) *as* $i..j$*; moreover, we abbreviate* $i..i$ *to* i*,* $0..\omega$ *to* $\star$ *and* $i{+}1..\omega$ *to* $>i$*. The lower bound of a multiplicity* $\mu \in \mathbf{M}$ *is denoted* $\lfloor \mu \rfloor$ *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* $\lfloor \mu \rfloor > 0$*.*
2. *We define a* subsumption ordering $\sqsupseteq \subseteq \mathbf{M} \times \mathbf{M}$ *over multiplicities, such that* $\mu_1 \sqsupseteq \mu_2$ *if* $\lfloor \mu_1 \rfloor \leq \lfloor \mu_2 \rfloor$ *and* $\lceil \mu_1 \rceil \geq \lceil \mu_2 \rceil$*.*

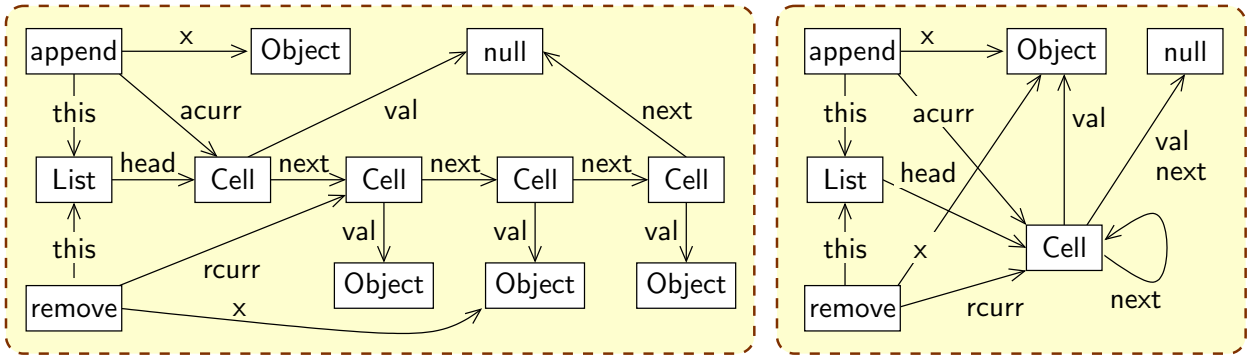

Figure 2: Instance and type graph representing a state of the program in Figure 1

3. *The* sum *of a set* $M \subseteq \mathbf{M}$ *is defined by* $\sum M = i..j$*, where* $i = \sum_{\mu \in M} \lfloor \mu \rfloor$ *and* $j = \sum_{\mu \in M} \lceil \mu \rceil$*.*
4. *A given set* X has multiplicity μ*, denoted* $X : \mu$*, if* $\lfloor \mu \rfloor \leq |X|$ *and* $|X| \leq \lceil \mu \rceil$*.*
5. *We select a collection of* base multiplities $\underline{\mathbf{M}} = \{0, 1, >\mathbf{1}\}$ *(chosen such 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.*

A shape graph is a graph with associated node and edge multiplicities. To be precise, for each node we have a multiplicity which expresses for how many "real" nodes it may stand; and for each combination of node and label we have two edge multiplicities, for the numbers of outgoing (resp. incoming) edges with that label from (resp. to) every instance of that node.

Definition 2.4 (shapes) *A* shape *is a tuple* $\mathcal{S} = \langle L, N, E, nd, in, out \rangle$*, in which* $\langle L, N, E \rangle$ *is a graph (sometimes denoted* $G_{\mathcal{S}}$*) and*

- $nd \colon N \to \mathbf{M}$ *is a node multiplicity function;*
- $out \colon N \to L \to \mathbf{M}$*, with* $\lceil out(p)(a) \rceil \leq \sum_{(p,a,q) \in E} \lceil nd(q) \rceil$ *for all* $p \in N$ *and* $a \in L$*, is an outgoing edge multiplicity function;*
- $in \colon N \to L \to \mathbf{M}$*, with* $\lceil in(q)(a) \rceil \leq \sum_{(p,a,q) \in E} \lceil nd(p) \rceil$ *for all* $p \in N$ *and* $a \in L$*, is an incoming edge multiplicity function.*

The class of shapes over L is denoted $\mathbf{Shape}_L$. We will use shape graphs as a richer, more expressive alternative to graph types. A typing $\tau \colon G \to T$ constrains G by *allowing* only certain nodes and edges in G, namely those that can be mapped to T; but it does not enforce any structure on G in the sense of *requiring* the presence of particular nodes and edges. In contrast, shapes can specify this type of requirement, by stating that a certain multiplicity is positive.

Definition 2.5 (shapings) *Given a graph* $G \in \mathbf{Graph}_L$ *and a shape* $\mathcal{S} \in \mathbf{Shape}_L$*, a* shaping *from* G *to* $\mathcal{S}$ *is a typing* $\tau \colon G \to G_{\mathcal{S}}$ *such that*

- $\{p \in N_G \mid \tau(p) = p'\} : nd(p')$ *for all* $p' \in N_{\mathcal{S}}$*;*
- $\{q \in N_G \mid (p, a, q) \in E_G\} : out(\tau(p))(a)$ *for all* $p \in N_G$ *and* $a \in L_G$*.*
- $\{p \in N_G \mid (p, a, q) \in E_G\} : in(\tau(q))(a)$ *for all* $q \in N_G$ *and* $a \in L_G$*;*

τ *is called* full *if it is, in addition, surjective on edges. We write* $\tau \colon G \to \mathcal{S}$ *to denote that* τ *is a shaping from* G *to* $\mathcal{S}$*; moreover,* $\tau : G \xrightarrow{\textbf{full}} \mathcal{S}$ *denotes that* τ *is full.*

We call a graph G an instance of a shape $\mathcal{S}$ if there exists a shaping $\tau \colon G \to \mathcal{S}$, and a *full instance* if there exists a full shaping $\tau : G \xrightarrow{\textbf{full}} \mathcal{S}$. For instance, Figure 3 shows two shapes; the left hand graph of Figure 2 is a full instance of both (with obvious shapings). Note that we depict the multiplicities by writing them at outgoing and incoming "ports" for each node, which combine all identically labelled edges. This is potentially confusing since it results in a placement of the multiplicities at the endpoints opposite from the usual in, e.g., UML. Furthermore, we have omitted multiplicities for the unary edges; in this paper, they will always equal 1.

For instance, the left hand shape in Figure 3 constrains the null-node multiplicity to 1: null is a value, of which there can be only a single instance. Furthermore, the incoming edge multiplicities of the head- and next–edges at the Cell-nodes are $\mathbf{0}..\mathbf{1}$, reflecting that Cell-nodes in a list may not be shared. The right hand shape specifies additional constraints: there are three Cell-nodes, with different incoming edges. Among other things, this rules out sharing of Cell-nodes among append- and remove-methods. Note that all multiplicities of the right hand graph are positive.

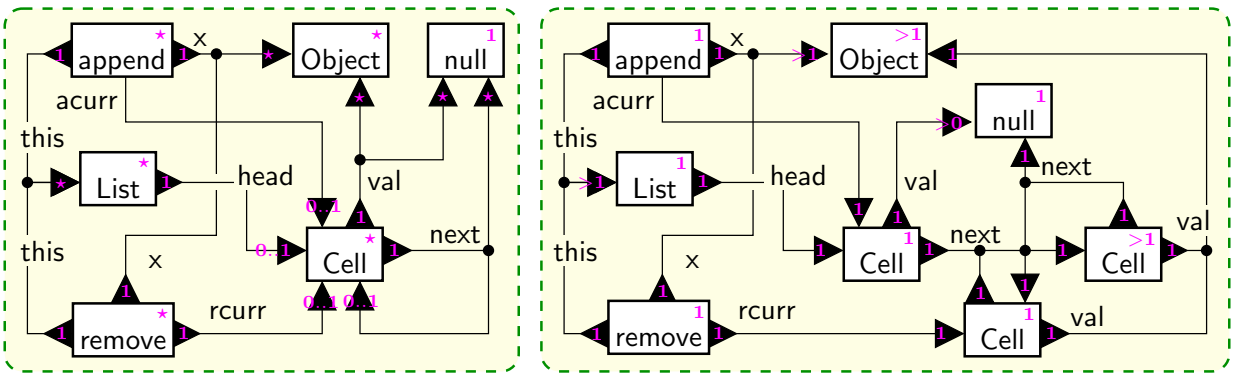

Figure 3: Two shapes for the (left hand) instance graph in Figure 2

Clearly, the structure imposed by a standard graph type T corresponds precisely to that imposed by a shape $\mathcal{S}$ with $G_{\mathcal{S}} = T$ and all multiplicities set to $\star$. On the other hand, if all node multiplicities are set to 1 and all edge multiplicities to the precise number of actual edges then a shape $\mathcal{S}$ has just its own underlying graph as an instance (up to isomorphism); and if all multiplicities are set to 0 then it only has the empty graph as an instance.

This suggests two standard ways to turn an ordinary graph G into a shape: by assigning the precise number of occurrences as multiplicities to all nodes and edges (which interprets G as an instance graph) or by setting all multiplicities to $\star$ (which interprets G as a type graph). Formally: if we define, for arbitrary $p \in N$ and $a \in L$

$$nd^{inst}(p) = 1 \qquad nd^{type}(p) = \star$$

$$out^{inst}(p)(a) = |\{q \in N \mid (p,a,q) \in E\}| \qquad out^{type}(p)(a) = \begin{cases} \star & \text{if } (p,a,q) \in E \text{ for some } q \\ 0 & \text{otherwise} \end{cases}$$

$$in^{inst}(p)(a) = |\{q \in N \mid (q,a,p) \in E\}| \qquad in^{type}(p)(a) = \begin{cases} \star & \text{if } (q,a,p) \in E \text{ for some } q \\ 0 & \text{otherwise} \end{cases}$$

then (clearly) $\mathcal{S}^{inst}(G) = \langle G, nd^{inst}, in^{inst}, out^{inst}\rangle$ and $\mathcal{S}^{type}(G) = \langle G, nd^{type}, in^{type}, out^{type}\rangle$ are shapes such that $\tau\colon G \to \mathcal{S}^{inst}(H)$ iff $G \cong H$ and $\tau\colon G \to \mathcal{S}^{type}(H)$ iff $\tau\colon G \to H$.

It is worth noting that shapes do not necessarily have instances. This is due to possible inconsistencies of node and edge multiplicity constraints: for instance, if there is an edge (p,a,q) with $nd(p) = 1$, $nd(q) = >\!\!\mathbf{1}$ and $out(p)(a) = in(q)(a) = 1$. We call a shape [fully] *satisfiable* if it has [full] instances. We will use $\mathcal{S}$ **fsat** to denote that $\mathcal{S} \in \mathbf{Shape}_L$ is fully satisfiable, and $\mathbf{fsat}(\mathcal{S})$ (with $\mathbf{S} \subseteq \mathbf{Shape}_L$) to denote the subset of $\mathcal{S} \in \mathbf{S}$ such that $\mathcal{S}$ **fsat**. We show in [9] that satisfiability is decidable; as a straightforward corollary, so is full satisfiability. Thus:

Theorem 2.6 $\mathcal{S}$ **fsat** *is decidable for arbitrary* $\mathcal{S} \in \mathbf{Shape}_L$.

Whereas shapings relate graphs to shapes, we may define relations directly between shapes, too. The following is the natural generalisation of graph morphisms.

Definition 2.7 (shape morphisms) *Given two shapes* $\mathcal{S}, \mathcal{R}$, *a* shape morphism *from* $\mathcal{S}$ *to* $\mathcal{R}$ *is a morphism* $\sigma\colon G_{\mathcal{S}} \to G_{\mathcal{R}}$ *such that*

- *For all* $p \in N_{\mathcal{R}}$, $nd_{\mathcal{R}}(p) \sqsupseteq \sum nd_{\mathcal{S}}(\sigma^{-1}(p))$;
- *For all* $p \in N_{\mathcal{S}}$ *and* $a \in L_{\mathcal{S}}$, $out_{\mathcal{R}}(\sigma(p))(a) \sqsupseteq out_{\mathcal{S}}(p)(a)$ *and* $in_{\mathcal{R}}(\sigma(q))(a) \sqsupseteq in_{\mathcal{S}}(q)(a)$.

We write $\sigma\colon \mathcal{S} \to \mathcal{R}$ *to denote that* σ *is a shape morphism from* $\mathcal{S}$ *to* $\mathcal{R}$.

Isomorphism of shapes is defined as usual. The following properties are immediate.

Proposition 2.8 *Let G be a graph and $\mathcal{S}, \mathcal{R}$ shapes.*

1. *$\phi: G \to H$ if and only if $\phi: \mathcal{S}^{type}(G) \to \mathcal{S}^{type}(H)$*
2. *$\tau: G \to \mathcal{S}$ if and only if $\tau: \mathcal{S}^{inst}(G) \to \mathcal{S}$.*
3. *If $\tau: G \to \mathcal{S}$ and $\sigma: \mathcal{S} \to \mathcal{R}$ then $\sigma \circ \tau: G \to \mathcal{R}$.*

In fact, we have the following result (where $\mathbf{Graph}_L$ is the category of L-graphs and graph morphisms and $\mathbf{Shape}_L$ that of L-shapes and shape morphisms):

Theorem 2.9 *For all L, $\mathcal{F}: \mathbf{Graph}_L \to \mathbf{Shape}_L$ with $\mathcal{S} \mapsto G_{\mathcal{S}}$ and $\sigma \mapsto \sigma$ is a functor, as is $\mathcal{U}: \mathbf{Shape}_L \to \mathbf{Graph}_L$ with $G \mapsto \mathcal{S}^{type}(G)$ and $\phi \mapsto \phi$; moreover, $\mathcal{F}$ and $\mathcal{U}$ form an adjunction.*

Proof sketch. The unit of this adjunction is given by the natural transformation $\eta: \mathbf{1} \to \mathcal{U}\mathcal{F}$ such that $\eta_{\mathcal{S}} = id_{N_{\mathcal{S}}}$ for all $\mathcal{S} \in \mathbf{Shape}$. □

3 Canonical shapes

Shapings are not unique: it is easy to construct examples where $\tau_i: G \to \mathcal{S}_i$ with $i = 1, 2$ for distinct τ_1, τ_2 (even if $\mathcal{S}1 = \mathcal{S}_2$). This is due to the absence of any constraints on duplication within shapes. We now single out a subclass of shapes which we call *canonical*, where such ambiguity is ruled out. For this purpose, we define a *similarity relation* on the nodes of a shape, and we then stipulate that canonical shapes may have no distinct similar nodes. Given a shape $\mathcal{S}$, we define $\sim_{\mathcal{S}} \subseteq N_{\mathcal{S}} \times N_{\mathcal{S}}$ as follows:

$$p \sim_{\mathcal{S}} q \text{ if and only if } out_{\mathcal{S}}(p) = out_{\mathcal{S}}(q) \text{ and } in_{\mathcal{S}}(p) = in_{\mathcal{S}}(q) \ . \tag{1}$$

(Recall that for all $p \in N_{\mathcal{S}}$, $out_{\mathcal{S}}(p)$ and $in_{\mathcal{S}}(p)$ are functions $L_{\mathcal{S}} \to N_{\mathcal{S}}$.) Thus, we count nodes as similar that have the same outgoing and incoming edge multiplicities. For instance, the nodes of the left hand side graph of Figure 3 are clearly dissimilar since they have distinct unary edges, whereas the three Cell-nodes of the right hand side graph are dissimilar because they have different sets of incoming edges. On the other hand, the last two Cell-nodes of the left hand graph in Figure 2 (or rather, its $\mathcal{S}^{inst}$-image) are similar.

Definition 3.1 *A shape $\mathcal{S}$ is* canonical *if*

- *For all $p \in N$, $nd(p) \in \underline{\mathbf{M}}^{>0}$;*
- *For all $(p, a, q) \in E$, $in(q)(a), out(p)(a) \in \underline{\mathbf{M}}^{>0}$.*
- *For all $p, q \in N$, $p \sim q$ implies $p = q$.*

A shaping $\tau: G \to \mathcal{S}$ is canonical if $\mathcal{S}$ is canonical and τ is full.

The class of canonical shapes over L is denoted $\mathbf{Canon}_L$. It follows that the left hand graph in Figure 3 and (the $\mathcal{S}^{inst}$-image of) the left hand graph in Figure 2 are not canonical: the multiplicities for the former are not in $\underline{\mathbf{M}}^{>0}$, whereas the latter has distinct similar nodes. On the other hand, the right hand graph of Figure 3 is canonical.

As discussed in the introduction, we propose to use canonical shapes as state abstractions. For this purpose we need a mapping (α in the introduction) from graphs to canonical shapes. This is guaranteed by the following theorem, which states that every graph has precisely one canonical shaping. For the proof see Appendix A.

Theorem 3.2 *For every graph G, there is precisely one canonical shape $\mathcal{S}$ (up to isomorphism) for which there exists a canonical shaping $\tau : G \xrightarrow{\textbf{full}} \mathcal{S}$; moreover, τ is unique in this respect.*

For the abstraction to be useful, it should result in a reduction of the size of the state space. A key insight is the following, proved in [9]:

Theorem 3.3 *For a given finite set of labels L, the set $\mathbf{Canon}_L$ of canonical shapes is also finite.*

The canonical shape of a graph G is obtained by partitioning the nodes of G according to the similarity relation defined in (1). That is, we first have to push similarity, which was defined for shapes, to graphs. This can be done using the functions out^{inst} and in^{inst} defined above, taking into account, however, that we only want base multiplicities, i.e., elements of $\underline{\mathbf{M}}$. To convert arbitrary multiplicities systematically to base multiplicities we use the following *factorisation* operation, where $\mu \in \mathbf{M}$, $M \subseteq \mathbf{M}$ and $f\colon X \to Y$:

$$\begin{aligned} \mu/M &= \{\nu \in M \mid \exists i : \lfloor \mu \rfloor \leq i \leq \lceil \mu \rceil \wedge \lfloor \nu \rfloor \leq i \leq \lceil \nu \rceil\} \\ f/M &= \{g\colon X \to Y/M \mid \forall x \in X : g(x) \in f(x)/M\} \end{aligned}$$

Thus, μ/M is the set of multiplicities in M that have an overlap with μ. Note that this also applies to "point multiplicities", i.e., where μ consists of a single integer only. For instance, $\star/\underline{\mathbf{M}} = \underline{\mathbf{M}}$, $>\!\mathbf{0}/\underline{\mathbf{M}} = 1..3/\underline{\mathbf{M}} = \{\mathbf{1}, >\!\mathbf{1}\}$ and $2/\underline{\mathbf{M}} = \{>\!\mathbf{1}\}$. f/M extends this to functions ranging over $\mathbf{M}$.

$$p \sim q \text{ if and only if } out^{inst}(p)/\underline{\mathbf{M}} = out^{inst}(q)/\underline{\mathbf{M}} \text{ and } in^{inst}(p)/\underline{\mathbf{M}} = in^{inst}(q)/\underline{\mathbf{M}} \ .$$

We then define $\mathcal{S}^{canon}(G) = \langle L, N^{canon}, E^{canon}, nd^{canon}, out^{canon}, in^{canon} \rangle$ with

$$\begin{aligned} N^{canon} &= N/\!\sim \\ E^{canon} &= \{([p]_\sim, a, [q]_\sim) \mid (p, a, q) \in E\} \\ nd^{canon} &= \{(P, |P|/\underline{\mathbf{M}}) \mid P \in N^{canon}\} \\ out^{canon} &= out^{inst}/\underline{\mathbf{M}} \\ in^{canon} &= in^{inst}/\underline{\mathbf{M}} \ . \end{aligned}$$

(Note that, strictly speaking, $|P|/\underline{\mathbf{M}}$, $out^{inst}/\underline{\mathbf{M}}$ and $in^{inst}/\underline{\mathbf{M}}$ are *sets*. However, it is easy to see that they are always singleton sets, which we equate here with the elements contained.) The following proposition states that this gives rise to a canonical shape for G. This also completes the proof of Theorem 3.2.

Proposition 3.4 *Let G be an arbitrary graph and define $\phi\colon N \to N/\!\sim$ by $\phi\colon p \mapsto [p]_\sim$ for all $p \in N$. Then $\mathcal{S}^{canon}(G)$ is a canonical shape with $\phi : G \xrightarrow{\textbf{full}} \mathcal{S}^{canon}(G)$.*

For instance, the right hand shape of Figure 3 is *not* the canonical shape of the left hand graph in Figure 2: the Object-nodes in the graph are not all similar and yet in the shape they are unified.

Canonising shapes. Alternatively, one may canonise arbitrary shapes. This can be useful for instance if, after manipulating a canonical shape (see Section 4), the resulting shape is not immediately canonical. Here, however, we run into the problem that an arbitrary shape $\mathcal{S}$ generally cannot be represented by a *single* canonical shape; in other words, it is generally not the case that all G for which $\tau\colon G \to \mathcal{S}$ have the same canonical shape. The best we can do is define a function $canon\colon \mathbf{Shape}_L \to \mathbf{2}^{\mathbf{Canon}_L}$ such that $canon(\mathcal{S})$ consists of all $\mathcal{S}^{canon}(G)$ (modulo isomorphism) for which $\tau\colon G \to \mathcal{S}$. (Note that this implies that $canon(\mathcal{S})$ only contains satisfiable shapes; in fact, if $\mathcal{S}$ itself is not satisfiable then $canon(\mathcal{S}) = \emptyset$.) We define *canon* through several transformations.

1. Normalise the multiplicities in $\mathcal{S}$ through $\_ \sqcap \underline{\mathbf{M}}$;
2. Widen the edge multiplicities from $\downarrow \underline{\mathbf{M}}$ to $\underline{\mathbf{M}}$;
3. Combine similar nodes by partitioning them through $\sim$ (see (1));
4. Widen the node multiplicities from $\downarrow \underline{\mathbf{M}}^{>0}$ to $\underline{\mathbf{M}}^{>0}$, analogous to the edge multiplicities.

To define these formally we need one more operation. Let $\mu \in \mathbf{M}, M \subseteq \mathbf{M}$ and $f\colon X \to Y$.

$$\begin{array}{lcl} \mu \sqcap M & = & \{\max(\lfloor\mu\rfloor, \lfloor\nu\rfloor)..\min(\lceil\mu\rceil, \lceil\nu\rceil) \mid \nu \in \mu/M\} \\ f \sqcap M & = & \{g\colon X \to Y \sqcap M \mid \forall x \in X : g(x) \in f(x) \sqcap M\} \ . \end{array}$$

Thus, $\mu \sqcap M$ is the set of all non-empty intersections of μ with multiplicities from M. For instance, $1..3 \sqcap \underline{\mathbf{M}} = \{1, 2..3\}$ and $2 \sqcap \underline{\mathbf{M}} = \{2\}$. We denote $\downarrow M = \{\mu \in \mathbf{M} \mid \exists \nu \in M : \nu \sqsupseteq \mu\}$ for the set of multiplicities subsumed by some multiplicity in M.

$$\begin{array}{lcl} \mathit{norm}: & \mathcal{S} \mapsto & \{\mathcal{R} \in \mathbf{Shape}_L \mid \mathcal{R}\ \mathbf{fsat}, \mathcal{R} \prec \mathcal{S}\} \\ \mathit{widen}_{\mathit{edge}}: & \mathcal{S} \mapsto & \{\langle L, N_\mathcal{S}, E_\mathcal{S}, \mathit{nd}_\mathcal{S}, f, g\rangle \mid f \in \mathit{out}_\mathcal{S}/\underline{\mathbf{M}}, g \in \mathit{in}_\mathcal{S}/\underline{\mathbf{M}}\} \\ \mathit{partition}: & \mathcal{S} \mapsto & \mathcal{S}/\!\sim \\ \mathit{widen}_{\mathit{node}}: & \mathcal{S} \mapsto & \{\langle L, N_\mathcal{S}, E_\mathcal{S}, f, \mathit{out}_\mathcal{S}, \mathit{in}_\mathcal{S}\rangle \mid f \in \mathit{nd}_\mathcal{S}/\underline{\mathbf{M}}\} \ . \end{array}$$

where $\mathcal{R} \prec \mathcal{S}$, expressing that $\mathcal{R}$ *refines* $\mathcal{S}$, is defined to hold if $L_\mathcal{R} = L_\mathcal{S}$ and

$$\begin{array}{rl} N_\mathcal{R} \subseteq & \{(p, f, g) \mid p \in N_\mathcal{S}, f \in \mathit{out}_\mathcal{S}(p) \sqcap \underline{\mathbf{M}}, g \in \mathit{in}_\mathcal{S}(p) \sqcap \underline{\mathbf{M}}\} \\ E_\mathcal{R} \subseteq & \{((p, f, g), a, (p', f', g')) \in N_\mathcal{R} \times L \times N_\mathcal{R} \mid (p, a, p') \in E_\mathcal{S}, f(a) \neq 0, g'(a) \neq 0\} \\ \mathit{nd}_\mathcal{R} \in & \bigcup \{h \sqcap \underline{\mathbf{M}}^{>0} \mid h\colon N_\mathcal{R} \to \mathbf{M}, \forall p \in N_\mathcal{S}\colon \mathit{nd}_\mathcal{S}(p) = \textstyle\sum_{(p,f,g) \in N_\mathcal{R}} h((p, f, g))\} \\ \mathit{out}_\mathcal{R} = & \{((p, f, g), f) \mid (p, f, g) \in N_\mathcal{R}\} \\ \mathit{in}_\mathcal{R} = & \{((p, f, g), g) \mid (p, f, g) \in N_\mathcal{R}\} \end{array}$$

and $\mathcal{S}/\!\sim = \mathcal{R}$ is defined by $L_\mathcal{R} = L_\mathcal{S}$ and

$$\begin{array}{rl} N_\mathcal{R} = & N_\mathcal{S}/\!\sim \\ E_\mathcal{R} = & \{([p]_\sim, a, [q]_\sim) \mid (p, a, q) \in E_\mathcal{S}\} \\ \mathit{nd}_\mathcal{R} = & \{([p]_\sim, \textstyle\sum_{q \sim p} \mathit{nd}_\mathcal{S}(q)) \mid p \in N_\mathcal{S}\} \\ \mathit{out}_\mathcal{R} = & \{([p]_\sim, \mathit{out}_\mathcal{S}(p)) \mid p \in N_\mathcal{S}\} \\ \mathit{in}_\mathcal{R} = & \{([p]_\sim, \mathit{in}_\mathcal{S}(p)) \mid p \in N_\mathcal{S}\} \ . \end{array}$$

Note that *norm* may blow up the size of the shape $\mathcal{S}$ exponentially, and also the number of normalised shapes may be exponential in the size of $\mathcal{S}$. We also use the (straightforward) extension to *sets* of shape graphs; moreover, if a transformation yields a singleton set we may equate this with the element contained therein. This gives rise to the following definition of *canon*:

$$\mathit{canon} = \mathit{widen}_{\mathit{node}} \circ \mathit{partition} \circ \mathit{widen}_{\mathit{edge}} \circ \mathit{norm} \ . \tag{2}$$

The canonisation mapping $\mathcal{S}^{\mathit{canon}}$ of graphs, defined above, can now be reformulated:

$$\mathcal{S}^{\mathit{canon}} = \mathit{widen}_{\mathit{node}} \circ \mathit{partition} \circ \mathit{widen}_{\mathit{edge}} \circ \mathcal{S}^{\mathit{inst}} \ . \tag{3}$$

(The multiplicity normalization phase can be skipped in this case because all multiplicities are positive point intervals to start with, and hence in $\downarrow \underline{\mathbf{M}}^{>0}$.) The result announced above is now formally stated by the following theorem, the proof of which can be found in Appendix A.

Theorem 3.5 *For all* $\mathcal{S} \in \mathbf{Shape}_L$, $\{[\mathcal{R}]_\cong \mid \mathcal{R} \in \mathit{canon}(\mathcal{S})\} = \{[\mathcal{S}^{\mathit{canon}}(G)]_\cong \mid \exists \tau\colon G \to \mathcal{S}\}$.

4 Future work

We have defined an abstraction from graphs to shapes. As outlined in the introduction, these results form a step in a trajectory that should eventually lead to a practically feasible abstraction technique for graph-based behavioural models. Two of the next steps should address transitions in this model and the question of property preservation. We briefly discuss the issues involved.

Graph transformations. As a basis of transitions between graphs we propose *graph transformations*. A graph transformation takes one graph to another while deleting some nodes and edges and creating others. To abstract from node identities, each graph transformation is accompanied by a partial one-to-one relation between nodes and edges in the source graph to those in the target graph, recording which elements are *preserved*, i.e., neither deleted nor created, by the transformation. This relation can either be given as a partial morphism from source to target graph or as a span of total, injective morphisms from a third, intermediate graph. The former corresponds to the so-called *single pushout* approach to graph transformations, the latter to the *double pushout* approach; for the purpose of this discussion the difference does not matter. Similar ideas can be fount in [1, 13].

Eventually, we are interested in characterising abstract transitions *without* referring to the concrete level, for otherwise they can still not be generated effectively. We therefore have to look more closely at how graph transformations are actually produced (the single or double pushout approaches) and lift this whole production process to the concrete level. We will investigate this in a future paper.

Graph properties. Various logics for expressing graph properties have been proposed recently; see, e.g., [1, 2, 3, 7, 10]. The important question for us is to what degree these are compatible with the canonical abstraction we have defined here. In particular, we want to have

$$\mathcal{S}^{canon}(G) \models^{\mathbf{Shape}} \psi \text{ implies } G \models^{\mathbf{Graph}} \psi \tag{4}$$

for a set of properties ψ that is as large as possible and can express interesting structural properties of program states. (The functions π, used in the introduction, that associate to each state the set of properties holding there, may be defined by $\pi(G) = \{\psi \mid G \models^{\mathbf{Graph}} \psi\}$ on the concrete level and $\pi(\mathcal{S}) = \{\psi \mid \mathcal{S} \models^{\mathbf{Shape}} \psi\}$ on the abstract level.) From the work of Sagiv et al. [12] we know that (for the similar though more dedicated models studied there) it is indeed possible to formulate such a satisfaction relation $\models^{\mathbf{Shape}}$, using essentially first-order predicate logic enriched with a predicate for connectivity.

Except for [1], the work cited above is concerned with properties of *individual states*. When used in a model for dynamically evolving behaviour, such properties essentially correspond to state invariants. This reduces the amount of information to be kept in the model: for instance, the relation between the nodes and edges of the source and target graphs of a transition, discussed above in the context of graph transformations, is superfluous in this setting. To express other types of properties, such as pre-/postconditions or temporal properties, one needs the additional ability to trace the identities of nodes over time. We have proposed such a logic, with a semantics defined over graphs (the relation $\models^{\mathbf{Graph}}$ in (4)) in [8], based on ideas about expressing dynamic (de)allocation previously developed in [4, 5]. It remains to be seen if the techniques of [12] can be used to provide a semantics over shapes that satisfies (4).

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] M. Benedikt, T. Reps, and M. Sagiv. A decidable logic for describing linked data structures. In S. D. Swierstra, editor, *Programming Languages and Systems — 8th European Symposium on Programming*, volume 1576 of *Lecture Notes in Computer Science*, pages 2–19. Springer-Verlag, 1999.

[3] L. Cardelli, P. Gardner, and G. Ghelli. A spatial logic for querying graphs. In P. Widmayer et al., editor, *Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 597–610. Springer-Verlag, 2002.

[4] D. Distefano, A. Rensink, and J.-P. Katoen. Model checking birth and death. In R. 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.

[5] D. Distefano, A. Rensink, and J.-P. Katoen. Who is pointing when to whom: On model-checking pointer structures. CTIT Technical Report TR–CTIT–03–12, Department of Computer Science, University of Twente, Sept. 2003.

[6] M. Huth, R. Jagadeesan, and D. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In D. Sands, editor, *Programming Languages and Systems — ESOP '01*, volume 2028 of *Lecture Notes in Computer Science*, pages 155–169. Springer-Verlag, 2001.

[7] 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.

[8] A. Rensink. Towards model checking graph grammars. In M. Leuschel, S. Gruner, and S. L. Presti, editors, *Proceedings of the* 3rd *Workshop on Automated Verification of Critical Systems*, Technical Report DSSE–TR–2003–2, pages 150–160. University of Southampton, 2003.

[9] A. Rensink. Canonical graph shapes. In *European Symposium on Programming*, Lecture Notes in Computer Science. Springer-Verlag, 2004. To appear.

[10] 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.

[11] 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.

[12] 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.

[13] D. Varró. Towards symbolic analysis of visual modeling languages. In P. Bottoni and M. Minas, editors, *International Workshop on Graph Transformation and Visual Modeling Techniques*, volume 72 of *Electronic Notes in Theoretical Computer Science*, 2002.

A Proofs of the theorems

Theorem 3.2 *For every graph G, there is precisely one canonical shape $\mathcal{S}$ (up to isomorphism) for which there exists a canonical shaping $\tau : G \xrightarrow{\textbf{full}} \mathcal{S}$; moreover, τ is unique in this respect.*

Proof. We show here the uniqueness of $\mathcal{S}$ and τ; their existence follows from Proposition 3.4 below. Assume canonical shapings $\tau_i \colon G \to \mathcal{S}_i$ for $i = 1, 2$; we show that there exists a unique isomorphism $\sigma \colon \mathcal{S}_1 \to \mathcal{S}_2$ and that $\tau_2 = \sigma \circ \tau_1$.

In fact, if we find an isomorphism $\sigma \colon \mathcal{S}_1 \to \mathcal{S}_2$ then it is necessarily unique. For suppose $\sigma' \colon \mathcal{S}_1 \to \mathcal{S}_2$ is also a isomorphism; then for an arbitrary $p \in N_1$ it follows that $out_2(\sigma(p)) = out_1(p) = out_2(\sigma'(p))$ and likewise for in; hence $\sigma(p) \sim \sigma'(p)$, implying $\sigma(p) = \sigma'(p)$ due to the canonicality of $\mathcal{S}_2$.

We show that τ_1 and τ_2 are injective on the same nodes and that they are surjective (on N_1 resp. N_2). It follows that $\sigma \colon N_1 \to N_2$ defined by $\tau_1(p) \mapsto \tau_2(p)$ for all $p \in N_G$ is a bijection and that $\tau_2 = \sigma \circ \tau_2$ (when regarding all as functions over node sets).

- Consider $p, p' \in N_G$ such that $\tau_1(p) = \tau_1(p') = p_1$. It follows that $\{q \in N_G \mid (p, a, q) \in E_G\} : out_1(p_1)(a)$ and $\{q \in N_G \mid (q, a, p) \in E_G\} : in_1(p_1)(a)$ for all $a \in L_G$; and likewise for p' instead of p. Note that $out_1(p_1)$ and $in_1(p_1)$ are completely fixed by this, due to the fact that they are required to be in $\underline{\mathbf{M}}$. By analogous reasoning, for $p_2 = \tau_2(p)$ and $p_2' = \tau_2(p')$ we have $out_2(p_2) = out_2(p_2')$ $(= out_1(p))$ and $in_2(p_2) = in_2(p_2')$ $(= in_1(p))$; hence $p_2 \sim p_2'$, which implies $p_2 = p_2'$ due to the canonicality of $\mathcal{S}_2$.

- Surjectivity of τ_1 (and, by symmetry, of τ_2) follows from the fact that $nd_1(p_1) \in \underline{\mathbf{M}}^{>0}$ for all $p_1 \in N_1$, and hence there is at least one $p \in N_G$ such that $\tau_1(p) = p_1$.

All that is left to do is demonstrating that σ is actually a shape morphism. For this purpose we show the following properties.

- $\sigma(e_1) \in E_2$ for all $e_1 \in E_1$. This is due to the surjectivity of τ_1, which implies the existence of some $e \in N_G$ such that $\tau_1(e) = e_1$; it follows by construction that $\sigma(e_1) = \tau_2(e) \in E_2$.

- $nd_2(\sigma(p_1)) = nd_1(p_1)$ for all $p_1 \in N_1$. To see this, note that $\{p \in N_G \mid \tau_1(p) = p_1\} = \{p \in N_G \mid \tau_2(p) = \sigma(p_1)\}$. Calling this set N, we have $N : nd_1(p_1)$ and $N : nd_2(\sigma(p_1))$, which implies the required equality since both node multiplicities are elements of $\underline{\mathbf{M}}$.

- $out_2(\sigma(p_1)) = out_1(p_1)$ and $in_2(\sigma(p_1)) = in_1(p_1)$ for all $p_1 \in N_1$ and $a \in L_1$. This follows from the argument on the similarity of the injectivity of τ_1 and τ_2, above. □

The proof of Theorem 3.5 relies on intermediate results for the auxiliary functions *norm*, $widen_{edge}$, *partition*, $widen_{node}$.

Lemma A.1 *Let $\mathcal{S}, \mathcal{S}_1, \mathcal{S}_2 \in \mathbf{Shape}_L$.*

1. *$norm(\mathcal{S})$ satisfies the following properties:*
   - *For all $\mathcal{R} \in norm(\mathcal{S})$, all node mulitplicities are in $\downarrow \underline{\mathbf{M}}^{>0}$, all edge multiplicities in $\downarrow \underline{\mathbf{M}}$, and there exists a morphism $\sigma_{\mathcal{R}} \colon \mathcal{R} \to \mathcal{S}$;*
   - *For all $G \in \mathbf{Graph}_L$, $\tau \colon G \to \mathcal{S}$ if and only if $\rho : G \xrightarrow{\textbf{full}} \mathcal{R}$ for some $\mathcal{R} \in norm(\mathcal{S})$ such that $\tau = \sigma_{\mathcal{R}} \circ \rho$.*
2. *Assume all edge multiplicities of $\mathcal{S}, \mathcal{S}_1, \mathcal{S}_2$ are in $\downarrow \underline{\mathbf{M}}$.*

- *$widen_{edge}(\mathcal{S}) = \{\mathcal{R}\}$ with all edge multiplicities of $\mathcal{R}$ in $\underline{\mathbf{M}}$, and there is a node multiplicity preserving morphism $\sigma_{\mathcal{S}}: \mathcal{S} \to \mathcal{R}$ that is an isomorphism from $G_{\mathcal{S}}$ to $G_{\mathcal{R}}$;*
- *If $\rho: \mathcal{S}_1 \to \mathcal{S}_2$ then $\rho: widen_{edge}(\mathcal{S}_1) \to widen_{edge}(\mathcal{S}_2)$ such that $\sigma_{\mathcal{S}_2} \circ \rho = \rho \circ \sigma_{\mathcal{S}_1}$.*

3. *Assume all edge multiplicities of $\mathcal{S}, \mathcal{S}_1, \mathcal{S}_2$ are in $\underline{\mathbf{M}}$.*

   - *$partition(\mathcal{S}) = \mathcal{R}$ such that $p \sim_{\mathcal{R}} q$ implies $p = q$, and there is an edge multiplicity preserving, surjective morphism $\sigma_{\mathcal{S}}: \mathcal{S} \to \mathcal{R}$ with $nd_{\mathcal{R}}(q) = \sum nd_{\mathcal{S}}(\sigma_{\mathcal{S}}^{-1}(q))$ for all $q \in N_{\mathcal{R}}$.*
   - *If $\rho: \mathcal{S}_1 \to \mathcal{S}_2$, then $\rho': partition(\mathcal{S}_1) \to partition(\mathcal{S}_2)$ such that $\sigma_{\mathcal{S}_2} \circ \rho = \rho' \circ \sigma_{\mathcal{S}_1}$.*

4. *Assume all node multiplicities of $\mathcal{S}, \mathcal{S}_1, \mathcal{S}_2$ are in $\downarrow \underline{\mathbf{M}}^{>0}$.*

   - *$widen_{node}(\mathcal{S}) = \{\mathcal{R}\}$ with all node multiplicities of $\mathcal{R}$ in in $\underline{\mathbf{M}}^{>0}$, and there is an edge multiplicity preserving morphism $\sigma_{\mathcal{S}}: \mathcal{S} \to \mathcal{R}$ that is an isomorphism from $G_{\mathcal{S}}$ to $G_{\mathcal{R}}$;*
   - *If $\rho: \mathcal{S}_1 \to \mathcal{S}_2$ then $\rho: widen_{edge}(\mathcal{S}_1) \to widen_{edge}(\mathcal{S}_2)$ such that $\sigma_{\mathcal{S}_2} \circ \rho = \rho \circ \sigma_{\mathcal{S}_1}$.*

Another auxiliary result is the following:

Lemma A.2 *If $\mathcal{S} \in \mathbf{Canon}_L$ and $\mathcal{R} \in \mathbf{Shape}_L$ such that all multiplicities in $\mathcal{R}$ are in $\underline{\mathbf{M}}$, then $\sigma: \mathcal{S} \to \mathcal{R}$ with σ surjective implies that σ is a shape isomorphism.*

We now come to the actual theorem.

Theorem 3.5 *For all $\mathcal{S} \in \mathbf{Shape}_L$, $\{[\mathcal{R}]_{\cong} \mid \mathcal{R} \in canon(\mathcal{S})\} = \{[\mathcal{S}^{canon}(G)]_{\cong} \mid \exists \tau: G \to \mathcal{S}\}$.*

Proof.

$\subseteq$ Let $\mathcal{R} \in canon(\mathcal{S})$; we construct $G \in \mathbf{Graph}_L$ and $\tau: G \to \mathcal{S}$ such that $\mathcal{S}^{canon}(G) \cong \mathcal{R}$. It follows that there are $\mathcal{S}_i = \langle L_i, N_i, E_i, nd_i, out_i, in_i \rangle$ for $i = 1, 2, 3$ such that

$$\begin{array}{l} \mathcal{S}_1 \prec \mathcal{S} \text{ such that } \mathcal{S}_1 \textbf{ fsat} \\ \mathcal{S}_2 = \langle L, N_1, E_1, nd_1, f, g \rangle \text{ with } f \in out_1/\underline{\mathbf{M}}, g \in in_1/\underline{\mathbf{M}} \\ \mathcal{S}_3 = \mathcal{S}_2/\!\sim \\ \mathcal{R} = \langle L, N_3, E_3, f, out_3, in_3 \rangle \text{ with } f \in nd_3/\underline{\mathbf{M}} \ . \end{array}$$

Now let G and τ_1 be such that $\rho : G \xrightarrow{\textbf{full}} \mathcal{S}_1$ — which exists due to $\mathcal{S}_1$ **fsat**. By Lemma A.1.1 it follows that $\tau: G \to \mathcal{S}$ for some τ. Furthermore, we may deduce

$\rho: \mathcal{S}^{inst}(G) \to \mathcal{S}_1$	due to Proposition 2.8.2
$\rho: widen_{edge}(\mathcal{S}^{inst}(G)) \to \mathcal{S}_2$	due to Lemma A.1.2
$\rho': partition(widen_{edge}(\mathcal{S}^{inst}(G))) \to \mathcal{S}_3$	due to Lemma A.1.3
$\rho': widen_{node}(partition(widen_{edge}(\mathcal{S}^{inst}(G)))) \to \mathcal{R}$	due to Lemma A.1.4

Moreover, ρ and ρ' are surjective in all contexts. Using (3) and Lemma A.2 it follows that ρ' is an isomorphism from $\mathcal{S}^{canon}(G)$ to $\mathcal{R}$.

$\supseteq$ Let $\tau: G \to \mathcal{S}$. Due to Lemma A.11 it follows that $\rho : G \xrightarrow{\textbf{full}} \mathcal{R}$ for some $\mathcal{R} \in norm(\mathcal{S})$. Using the same line of reasoning as above we may conclude $\mathcal{S}^{canon}(G) \cong \mathcal{R}'$ for $\mathcal{R}' \in widen_{node}(partition(edgewiden(\mathcal{R}))) \subseteq canon(\mathcal{S})$. $\square$