Towards A Shape Analysis for Graph Transformation Systems

Dominik Steenken, Heike Wehrheim, Daniel Wonisch

Universität Paderborn
Institut für Informatik
33098 Paderborn, Germany
{dominik,wehrheim,dwonisch}@mail.uni-paderborn.de

Abstract. Graphs and graph transformation systems are a frequently used modelling technique for a wide range of different domains, covering areas as diverse as refactorings, network topologies or reconfigurable software. Being a formal method, graph transformation systems lend themselves to a formal analysis. This has inspired the development of various verification methods, in particular also model checking tools.

In this paper, we present a verification technique for infinite-state graph transformation systems. The technique employs the abstraction principle used in shape analysis of programs, summarising possibly infinitely many nodes thus giving shape graphs. The technique has been implemented using the 3-valued logical foundations of standard shape analysis. We exemplify the approach on an example from the railway domain.

1 Introduction

Graph transformation systems (GTSs, [CMR+97]) have - in particular due to their visual appeal - become a widely used technique for system modelling. They are employed in numerous different areas, ranging from the specification of visual contracts for software to dynamically evolving systems. They serve as a formally precise description of the behaviour of complex systems. Often, such systems are operating in safety critical domains (e.g. railway, automotive) and their dependability is of vital interest. Hence, a number of approaches for the analysis of graph transformation systems have been developed [Tae03], in particular also model checking techniques [RSV04, SV03, Ren03, BCK08, BBKR08, SWJ08].

Model checking allows to fully automatically show properties of system models, for instance for properties specified in temporal logic. Model checking proceeds by exploring the whole state space of a model, i.e. in case of graph transformation systems by generating the set of graphs which are reachable from a given start graph by means of rule application. While existing tools have proven to be able to tackle also large state spaces, standard model checking techniques fail when the state space becomes infinite.
There are, in general, two approaches to dealing with very large or even infinite state spaces. The first approach is to devise a clever way of selecting a finite subset of states which is sufficient for proving the desired properties, effectively constructing an under-approximation of the system. This concept is explored e.g. in bounded model checking [BCC+03]. The second approach is to construct an abstraction, i.e. a finite representation of a superset of the state space. This over-approximation of the system is then used to show certain properties of the original system.

In this paper, we propose an new approach towards a verification technique for infinite state graph transformation systems using over-approximation. The technique follows the idea of shape analysis algorithms for programs [SRW02] which are used to compute properties of a program’s heap structures. Shape analyses compute abstractions of heap states by collapsing certain sets of identical nodes into so-called summary nodes. Thereby, an infinite number of heap states can be finitely represented. Such shapes can be used to derive structural properties about the heap.

This principle of summarisation in GTSs has already been presented in numerous other works, for example Rensink et al. [Ren04, RD06] or Bauer et.al. [BBKR08] which introduce so-called abstract graph transformations. Our approach is set apart from these by strict adherence to the formalism presented in [SRW02], which immensely simplifies implementation and gives us a level of parametrization that other approaches lack. A more thorough discussion of advantages of our approach over related work will be presented in section 6.

Here, we present a shape analysis for GTSs which is directly based on the 3-valued logical foundations of standard shape analysis. Given this logical basis for shape graphs, we define rule application on shape graphs via a constructive definition of materialisation and summarisation. The technique can thus be directly implemented as defined, even re-using parts of the logical machinery of TVLA [BLARS07], the most prominent shape analysis tool. In order to illustrate our technique, we exemplify it on a simple GTS model from the railway domain.

The paper is structured as follows. The next section will give the basic definitions for our approach. Section 3 introduces materialisation and summarisation on graphs and thereby defines the application of rules on shape graphs. Section 4 shows the correctness of our approach, i.e. shows that by rule application on shape graphs an overapproximation of the set of reachable graphs is computed. The next section then reports on the implementation. Finally, Sect. 6 concludes, further discusses related work and gives some directions for future research.
2 Background

This section introduces the basic definitions that are required to formulate our main results. In order to illustrate the definitions in this section, we use the following example.

A rail network is given by a set of stations (S) and a set of rail sections, called tracks (T), which are connected by a non-reflexive, antisymmetric relation called “next”. A number of vehicles, called “RailCabs” (RC), traverse this rail network to transport passengers (P) from one station to another. The current position of each RailCab is described by a non-reflexive, antisymmetric and injective relation called “on”, which connects each RailCab with exactly one track or station. This relation is also used to describe the position of a passenger and can connect a passenger with either a station or a RailCab.

Since RailCabs and passengers may enter or leave the rail network at any time, the number of possible states is infinite, even if the rail network itself is finite. In the following, we will first describe how such a system can be modeled using a graph transformation system. Then we will show how to transform that graph transformation system into first-order logic.

2.1 Graphs and Graph Transformation Systems

For the sake of completeness, we begin by defining graphs and morphisms.

![Fig. 1: A very simple rail network](image)

**Definition 1.** A graph $G$ is a pair $(N, E)$, where $N$ is a set of nodes and $E \subseteq N \times \mathcal{L} \times N$ is a set of labeled edges for some label set $\mathcal{L}$. For any graph $G$, $N_G$ and $E_G$ denote its node and edge sets, respectively.
This definition restricts the class of graphs we are considering to those in which no more than a single edge with a given label may exist between any two nodes. The generic concept of a morphism extends to these graphs in a natural way.

**Definition 2.** For graphs $G$ and $H$, a morphism $f: G \rightarrow H$ is a function $f : N_G \rightarrow N_H$ extended to edges by $f((n, l, n')) = (f(n), l, f(n'))$ such that $f(E_G) \subseteq E_H$.

Figure 1 shows a graph representing a very simple rail network consisting of two stations which are connected by two tracks such that the resulting structure forms a cycle. Note that we include a simple notion of typing in the graph. The type of a node is represented by a loop labelled with the name of the type. Such type loops are not displayed as edges but rather as part of the node name. Thus, instead of displaying a self-edge of $r_1$ labelled “RC”, we label the node $r_1 : RC$.

In order to model the dynamic behaviour of a system represented by a graph, we need to transform graphs into other graphs. For this, graph production rules can be used. In this paper, we take an operational, not categorical, view on graph transformation. As a consequence, we favor a simple approach to graph production rules, as the following definitions show.

**Definition 3.** A graph production rule $P = (L, R)$ consists of two graphs $L$ and $R$ called the left hand side and the right hand side, respectively.

Figure 2 shows an example of a rule which describes the simplest way to model the traversal of a track. Here, we use node names instead of injective morphisms to identify nodes appearing in the left as well as right hand side. Aside from this technicality we use the standard SPO approach to rule definition and application [CMR+97].

In order to make node creation / deletion explicit, we use the following sets:

$$
N^- = N_L \setminus N_R, \quad E^- = E_L \setminus E_R \quad \text{(deleted nodes and edges)}
$$

$$
N^+ = N_R \setminus N_L, \quad E^+ = E_R \setminus E_L \quad \text{(created nodes and edges)}
$$
These sets are used to define the effect of an application of a production rule on a graph $G$.

**Definition 4.** Let $P$ be a production rule, $G$ a graph. The rule $P = \langle L, R \rangle$ can be applied on $G$ if we can find an injective morphism $m : L \to G$ (called a matching). If $m$ is a matching, then the application of $P$ onto $G$ with matching $m$ is the graph

$$H = (N_H, E_H) \text{ with }$$
$$N_H = (N_G \setminus m(N^-)) \cup N^+$$
$$E_H = ((E_G \setminus m(E^-)) \cup \hat{m}(E^+)) \cap (N_H \times \mathcal{L} \times N_H)$$

where $\hat{m} = m \cup \text{id}_{N^+}$.

For this production application we write $G \xrightarrow{P,m} H$. Similarly, $G \xrightarrow{P} H$ holds if there is some $m$ such that $G \xrightarrow{P,m} H$ and $G \to H$ if there is furthermore a production rule $P$ such that $G \xrightarrow{P} H$. We let $\to^*$ denote the transitive and reflexive closure of $\to$. With these definitions at hand, we can define the set of reachable graphs of a graph transformation system.

**Definition 5.** A graph transformation system $GT = (G_0, (P_i)_{i \in I})$ consists of a start graph $G_0$ and a set of production rules $P_i$, $i \in I$. The set of reachable graphs of a graph transformation system $GT$ is

$$\text{reach}(GT) = \{G \mid G_0 \xrightarrow{\to^*} G\}$$

In this paper we are interested in proving properties of the set of all reachable graphs. An example of this is the Covering Problem where the objective is to decide whether the graph transformation system can produce a graph which includes a given “forbidden” graph as a subgraph.

### 2.2 3-valued Logic

In our approach, graphs are represented by logical predicates over sets of individuals, or, more precisely, by interpretations of such predicates. To clarify this, we present the following definitions, which closely follow those given in [SRW02].

First-order logical formulae, consecutively referred to simply as formulae, over a set $\mathcal{V}$ of variables and a set $\mathcal{P}$ of predicate symbols are defined using the well-known inductive definition. Since a formula in itself has no real meaning, it must be interpreted in order to assign a truth value to it.
Definition 6. An interpretation of a set of predicate symbols $\mathcal{P}$ is a function $\iota$ that assigns each predicate symbol a function over a “domain set” or “universe” $U$, mapping to a set $\mathcal{T}$ of truth values (e.g. $\{0, 1\}$ for boolean logic).

$$\iota : \mathcal{P}_k \rightarrow (U^k \rightarrow \mathcal{T})$$

Here, $\mathcal{P}_k \subset \mathcal{P}$ describes the set of $k$-ary predicate symbols.

While this assigns “meaning” to the predicate symbols with respect to some domain set $U$, the formula may not yet be evaluable since not all its variables may be bound by a quantifier. To deal with these free variables, assignments are used.

Definition 7. Let $\text{free}(\varphi) \subset \mathcal{V}$ be the set of free variables of a formula $\varphi$. An assignment is a function which maps $\mathcal{V}$ to individuals from a domain set $U$.

$$m : \text{free}(\varphi) \rightarrow U$$

An assignment is called complete for $\varphi$ if it is total over $\text{free}(\varphi)$.

We now want to encapsulate the concept of the “meaning” of formulae, as it exists separate from any particular formula. By doing so we construct a consistent framework with respect to which a set of distinct formulae can be evaluated. We achieve this by employing the mathematical concept of a logical structure.

Definition 8. A logical structure $S = \langle U, \mathcal{P}, \iota \rangle$ is a tuple consisting of a set of individuals $U$, constituting the “domain” on which the predicates operate, a set of predicate symbols, which essentially “name” predicates over $U$, and an interpretation $\iota$, mapping each predicate symbol to an actual predicate over $U$. For the sake of simplicity, $\langle U, \mathcal{P}, \iota \rangle$ can be shortened to $\langle U, \iota \rangle$.

A logical structure is called $n$-valued, if for the target set $\mathcal{T}$ of the predicates $|\mathcal{T}| = n$ holds. In order to formulate important distinctions between predicates, we partition the set $\mathcal{P}$ into the sets of so-called core predicates $\mathcal{C}$ and instrumentation predicates $\mathcal{I}$. Later on, $\mathcal{C}$ will encode basic properties, while $\mathcal{I}$ will be used to increase the precision of the analysis with respect to a given property. The set $\mathcal{C}$ is further subdivided into the set of unary core predicates $\mathcal{C}^1$ and the set of binary core predicates $\mathcal{C}^2$. $k$-ary predicates with $k > 2$ are not required for the graph encodings used in this paper.

In order to represent graphs and shape graphs using first-order predicate logic, we make use of the concept of 3-valued logic. The truth values contained in $\mathcal{T}$ are 0 (false), 1 (true) and $\frac{1}{2}$ (maybe). Since the meaning of the logical connectives and operators are not clear in the context of 3-valued logic, their meaning must be explicitly defined. Before we can do this, we need to relate the elements of $\mathcal{T}$ to each other in terms of a logical order.
Definition 9. For \( l_1, l_2 \in \mathcal{T} = \{0, \frac{1}{2}, 1\} \), the logical order \( \leq \) on truth values is defined as follows:

\[
l_1 \leq l_2 \iff (l_1 = l_2) \\
v \lor l_1 = 0 \\
v \lor l_1 = \frac{1}{2} \land l_2 = 1
\]

With respect to this ordering, we can also define the operations \( \min, \max : \mathcal{P}(\mathcal{T}) \to \mathcal{T} \) with the usual semantics. Furthermore, we define the complement operator \( 1^- \) by

\[
1^-0 = 1 \\
1^-\frac{1}{2} = \frac{1}{2} \\
1^-1 = 0
\]

Having thus specified how the truth values in \( \mathcal{T} \) relate to each other, we can now define the semantics of 3-valued logic.

Definition 10. The meaning of formulae \( \varphi \) in a 3-valued logical structure \( S = \langle U, \mathcal{P}, \iota \rangle \), according to an assignment \( m \), denoted by \( \llbracket \varphi \rrbracket^S_m \), is inductively defined as follows:

Atoms  For \( l \in \mathcal{T} \):

\[
\llbracket l \rrbracket^S_m = l
\]

Predicates  For \( p \in \mathcal{P}_k \) with free variables \( x_1, \ldots, x_k \):

\[
\llbracket p(x_1, \ldots, x_k) \rrbracket^S_m = \iota(p)(m(x_1), \ldots, m(x_k))
\]

Connectives  For some formulae \( \varphi, \psi \):

\[
\llbracket \varphi \land \psi \rrbracket^S_m = \min \left( \llbracket \varphi \rrbracket^S_m, \llbracket \psi \rrbracket^S_m \right)
\]

\[
\llbracket \varphi \lor \psi \rrbracket^S_m = \max \left( \llbracket \varphi \rrbracket^S_m, \llbracket \psi \rrbracket^S_m \right)
\]

Negation  For a formula \( \varphi \):

\[
\llbracket \neg \varphi \rrbracket^S_m = 1^- \llbracket \varphi \rrbracket^S_m
\]

Quantifiers  For a formula \( \varphi \):

\[
\llbracket \forall x_1 : \varphi \rrbracket^S_m = \min_{u \in U} \llbracket \varphi \rrbracket^S_{m[x_1 \mapsto u]}
\]

\[
\llbracket \exists x_1 : \varphi \rrbracket^S_m = \max_{u \in U} \llbracket \varphi \rrbracket^S_{m[x_1 \mapsto u]}
\]
With 3-valued logic now fully defined, we aim to obtain a notion of “graph abstraction” from it. Ideally, we would like to be able to construct abstract graphs from more concrete graphs by using single nodes to represent possibly infinitely large sets of nodes, and unambiguously define a kind of “generalisation relation”, ordering graphs by their level of abstraction. Since we will represent graphs by 3-valued logical structures, this translates directly to the need for a generalisation relation between logical structures. In the following, we will define such a relation.

In order to achieve this, we firstly require a notion which distinguishes “abstract” truth values \(\frac{1}{2}\) from less abstract ones \(0, 1\).

**Definition 11.** For \(l_1, l_2 \in T = \{0, \frac{1}{2}, 1\}\), the information order \(\sqsubseteq\) on truth values is defined as follows:

\[
l_1 \sqsubseteq l_2 \iff (l_1 = l_2) \lor (l_2 = \frac{1}{2})
\]

Intuitively, \(x \sqsubseteq y\) means that \(y\) is “more abstract”, i.e. contains less information, than \(x\). The next step is to extend this concept to logical structures. This requires the addition of two mandatory predicate symbols to the logical structures.

The first and most important of these is the summary predicate \(\text{sm}\). Interpretations of this predicate are restricted to the values \(0, \frac{1}{2}\). If it is \(\frac{1}{2}\) for any individual \(u\), this means \(u\) may or may not stand for a whole set of nodes. If it is \(0\) for \(u\), then \(u\) is guaranteed to represent a single individual.

The second predicate is \(=\). This binary predicate is imbued with the following semantics in the context of 3-valued logic. Let \(x_1, x_2\) be two free variables. Then:

\[
\llbracket x_1 = x_2 \rrbracket^S_m = \begin{cases} 
0 & \text{if } m(x_1) \neq m(x_2) \\
1 & \text{if } m(x_1) = m(x_2) \land \iota(sm)(m(x_1)) = 0 \\
\frac{1}{2} & \text{otherwise}
\end{cases}
\]

Using these two new predicates, we now obtain the notion of embedding by the following definition.

**Definition 12.** Let \(S = \langle U, \mathcal{P}, \iota \rangle, S' = \langle U', \mathcal{P}', \iota' \rangle\) be two logical structures and \(f : U \to U'\) be a surjective function. We say that \(f\) embeds \(S\) in \(S'\) iff \(\forall k \forall p \in \mathcal{P}_k, \forall u_1, \ldots, u_k \in U\)

\[
\iota(p)(u_1, \ldots, u_k) \subseteq \iota'(p)(f(u_1), \ldots, f(u_k)) \tag{1}
\]

and \(\forall u' \in U'\)

\[
(|\{u | f(u) = u'\}| > 1) \sqsubseteq \iota'(sm)(u') \tag{2}
\]

We write \(S \sqsubseteq S'\) to denote that there is some surjective function \(f\) which embeds \(S\) in \(S'\).
\[ U = \{r, p, s_1, s_2, t_1, t_2\} \]
\[ C^1 = \{RC, T, S, sm\} \]
\[ C^2 = \{on, next\} \]

<table>
<thead>
<tr>
<th>[ i \in (C^1) ]</th>
<th>[ RC ]</th>
<th>[ T ]</th>
<th>[ S ]</th>
<th>[ sm ]</th>
<th>[ i (on) ]</th>
<th>[ r_1 ]</th>
<th>[ r_2 ]</th>
<th>[ s_1 ]</th>
<th>[ s_2 ]</th>
<th>[ t_1 ]</th>
<th>[ t_2 ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ r_1 ]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[ r_2 ]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[ s_1 ]</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>[ s_2 ]</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>[ t_1 ]</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>[ t_2 ]</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: A sample graph

Again, intuitively, \( S \sqsubseteq S' \) means that \( S' \) is in some way a “generalisation” of \( S \). Thus, by means of the embedding function \( f \), \( S' \) “represents” \( S \). Since there may be infinitely many other logical structures which embed into \( S' \), the term generalisation is appropriate. Aside from the other results we will show later in this paper, the embedding relation can be used to determine whether two logical structures are isomorphic.

**Definition 13.** Two logical structures \( S_1 = \langle U_1, P_1, \iota_1 \rangle \) and \( S_2 = \langle U_2, P_2, \iota_2 \rangle \) are isomorphic, iff there exists a bijective function \( f : U_1 \to U_2 \) such that for all \( k \forall p \in P_k \forall u_1, \ldots, u_k \):

\[ \iota_1 (p) (u_1, \ldots, u_k) = \iota_2 (p) (f (u_1), \ldots, f (u_k)) \]

It can be shown that this is equivalent to \( (S_1 \sqsubseteq S_2) \land (S_1 \sqsupseteq S_2) \).

### 2.3 Graph and Shape Graph Encodings

Using the definitions of a logical structure given above, we can now encode arbitrary graphs. The set of nodes \( N \) of a graph will be represented by the domain set \( U \). The edge labels \( L \) will give us the set of predicate symbols \( P \). The edges for each label in \( L \) will be encoded in the interpretation of the predicate symbols in \( P \), given by \( \iota \).

**Definition 14.** Let \( G \) be a graph. The 2-valued encoding of \( G \), denoted \( ls (G) \), is a 2-valued logical structure \( S = \langle U, P, \iota \rangle \) with \( U = N_G \), \( C^1 \cup C^2 = P = L \) and \( \iota \) defined by

- For \( p \in C^2 \): \( \iota (p) (u_1, u_2) = 1 \iff (u_1, p, u_2) \in E_G \)
\[ U = \{ r, p, s, t, sm \} \]
\[ C^1 = \{ RC, T, S, sm \} \]
\[ C^2 = \{ on, next \} \]

\[
\begin{array}{c|cccc|c|c|c|c|c|c}
\iota \ (C^1) & RC & T & S & sm & \iota \ (on) & r & s & \iota \ (next) & r & t \\
r & 1 & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\
p & 0 & 1 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\
s & 0 & 0 & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\
t & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\
\end{array}
\]

Fig. 3: A sample shape graph

- For \( p \in C^1: \iota \ (p) \ (u) = 1 \iff (u, p, u) \in E_G \)
- For \( sm: \iota \ (sm) \ (u) = 0 \)

Table 1 shows an encoding of the graph presented in Fig. 1. A shape graph is then simply represented by a 3-valued logical structure. Figure 3 shows an example of a shape graph. Individuals for which the summary predicate evaluates to \( \frac{1}{2} \) are represented by dashed nodes, while other unary and binary predicates which evaluate to \( \frac{1}{2} \) are shown as dashed lines. As mentioned in Sect. 2.2, shape graphs which contain summarised nodes may represent an infinite number of concrete graphs via the embedding relation. In the following we will use the terms (concrete) graph and 2-valued logical structure, as well as shape graph and 3-valued logical structure synonymously.

Now let us assume that we are given a graph \( G \), and construct the corresponding 2-valued logical structure \( \langle U, P, \iota \rangle = S := ls \ (G) \). If we are given a formula \( f \) which uses only predicates contained within \( P \), we can determine whether \( f \) is satisfiable in \( S \) or not. Translated back into graph terminology, the formula represents a graph pattern that \( G \) either matches or not.

Consider for example the formula \( \varphi = n \ (x_1, x_2) \land n \ (x_2, x_3) \land n \ (x_3, x_1) \). If \( \varphi \) is satisfiable in \( S \), this means that the graph \( G \) contains at least one triangle formed by edges labelled “n”.

\[
\text{Table 1}
\]
3 Methods

In this section we present the methods used in our verification algorithm. For illustration, we also introduce a simple reconfigurable mechatronic system as a running example.

When applying shape analysis techniques on graph transformation systems, we look at so-called shape graph transformation systems (STSs) instead of graph transformation systems. A shape graph transformation system is quite similar to a graph transformation system: it consists of a starting (shape) graph and a set of (shape) production rules. Shape graphs were defined in Sect. 2.3 and shape productions will be defined in Sect. 3.3. For now, we can assume shape productions of an STS to be the same as the classical production rules of a GTS. The main difference between STSs and GTSs is then, how productions are applied. We split the application of productions into two steps: a materialisation step and an application step.

The application step basically works the same as before: with respect to some matching, add the new nodes and edges and remove the deleted nodes and edges as indicated by the production. However, to be able to do this safely, we need an injective matching from the nodes of the left hand side of the productions to some nodes of the shape graph. On the other hand, we might want to allow non-injective matchings to respect the fact that we may have summary nodes in the shape graph. In order to satisfy both constraints, we introduce the pre-step of “materialisation”\(^1\). The materialisation step materialises the left hand side of the production into the shape graph such that the left hand side of the production is explicitly present in the resulting shape graph. After that an injective matching can easily be found.

Figure 4 shows the shape graph transformation system of our running example. The starting shape graphs of the STS describes a situation where two railroad stations are connected by a number of tracks. Furthermore, there are two RailCabs, which are initially positioned on one of the tracks. In addition to the predicates already introduced in Sect. 2.1, we use the unary predicate empty to indicate the absence of any RailCab on a given track. The starting shape graph also makes use of an instrumentation predicate called is_colliding, which will be discussed later.

The example STS contains three (shape) production rules: MoveSingle, EnterStation, and LeaveStation. The MoveSingle production allows the RailCabs to move from one track to another empty track. The EnterStation production allows a RailCab to move from a track into a railway station. Finally, the LeaveStation production allows the RailCab to move out of the station onto an empty track.

Since we are illustrating a verification algorithm, we also need a property to verify. For this, a “forbidden pattern”, represented by a 2-valued logical struc-

\(^1\) The term “materialisation” is taken from [SRW02].
Fig. 4: Shape graph transformation system describing a simple reconfigurable mechatronic system.
Fig. 5: Forbidden pattern. Describes the situation that two RailCabs are positioned on the very same track.

...
the materialisation led to a valid shape graph and to sharpen the final result of the production application. Thus, more precisely, a production application in a shape graph transformation systems consists of four steps: 1. materialisation, 2. coerce, 3. application, and 4. coerce. The following subsections explain each of the steps in detail.

### 3.1 Materialisation

In classical graph transformation systems, a rule \( P = \langle L, R \rangle \) is applied to a graph \( G \) by first finding a matching \( m \) mapping \( L \) into \( G \). Since \( G \) may be a shape graph in our approach, summary nodes and \( \frac{1}{2} \)-valued edges may hide possible matchings. Thus, the graph needs to be transformed before the rule can be applied. The first step of this procedure is called materialisation.

First of all, in order to be able to talk about rule application using first-order logic, we need to transform the left-hand side of a rule into a formula which is satisfiable in a given logical structure if and only if the rule is applicable to the graph represented by the logical structure.

**Definition 15.** Let \( P = \langle L, R \rangle \) be a graph production rule. The production formula \( \varphi_P \) corresponding to \( P \) is given by

\[
\varphi_P = \bigwedge_{(n, l, n') \in E_L} l(n, n') \land \bigwedge_{(n, l, n) \in E_L} l(n) \land \bigwedge_{n_1, n_2 \in N_L} \neg (n_1 = n_2) \land \bigwedge_{n \in N_L} \neg sm(n)
\]

The \( \neq \) here means the non-equality of the variable symbols, while \( = \) is a regular predicate.

It is easy to see that this formula satisfies the condition mentioned above. Now we need to transform the graph. Intuitively, we would want something like this

\[
\text{focus}_P(S) = \{ S' \mid S' \subseteq S \land \varphi_P \text{ is satisfiable in } S' \}
\]

meaning all possible graphs which are embeddable in \( S \) and to which the rule can be applied. Unfortunately, this set can be infinitely large. Thus, we only compute a set \( \text{mat}_P(S) \) such that each element in \( \text{focus}_P(S) \) can be embedded in at least one element from \( \text{mat}_P(S) \).

In order to construct the set \( \text{mat}_P(S) \), let us now assume that we have a graph \( G \), its corresponding logical structure \( \langle N_G, L, \iota \rangle = S := \text{ls}(G), \) a production rule \( P = \langle L, R \rangle \), and a matching \( m : L \to N_G \) which gives rise to an assignment \( \bar{m} : \text{free}(\varphi_P) \to N_G \). Let \( N_G^{\text{sum}} \) be the set of summary nodes in \( G \).

We observe that the injectivity terms of \( \varphi_P \) (see Def. 15) do not evaluate to 0 if two nodes in \( L \) are mapped to the same node, if that node is a summary
node. Furthermore we see, taking into account the definition of \(=\) in our 3-valued logical structures, that \(\varphi_P\) could only evaluate to \(0\) if some nodes or edges required by the rule are \textit{definitely} not there, meaning that in all other cases we can not exclude the possibility that \(L\) may be applicable. We thus call \(P\) \textit{potentially applicable} with matching \(m\), if \([\varphi_P]_m^S \neq 0\). This generalises the notion of applicability to shape graphs, represented by 3-valued logical structures.

Let \(k(v) := |\{n \in L \mid m(n) = v\}|\) be the number of nodes in \(L\) that are mapped to a given node \(v \in N_G\) by \(m\). If \(v \in m(L), \iota^S(sm)(v) = 0\) and \([\varphi_P]_m^S \neq 0\), then the definition of \(\varphi_P\) guarantees that \(k(v) = 1\). If \(sm(v) = \frac{1}{2}\) however, \(k(v)\) may be greater than 1. In order to apply the rule, we must match each node in \(L\) to a non-summarised node in \(G\). This means that we must “materialise” \(k(v)\) nodes out of every summary node \(v\) in \(\Gamma(m)\). Since we do not know whether \(v\) represented \(k(v)\) nodes or more, we have to account for both possibilities. That is, we need to construct a graph for the case that it represented exactly \(k(v)\) nodes, meaning that \(v\) itself will vanish from the graph, and another graph for the case that there are still more nodes represented by \(v\), in which case \(v\) will remain. Predicate valuations will be raised to \(1\) where the rule requires this and derived from their valuations in the shape graph where they do not intersect with the rule. The following definition summarises this in a concise fashion.

**Definition 16.** Let \(G\) be a graph, \(S = ls(G) = \langle U, P, \iota \rangle\), \(P = \langle L, R \rangle\) be a production rule and \(M = \{ m \mid [\varphi_P]_m^S = \frac{1}{2} \}\). Let \(\Gamma(m) := m(N_L) \cap N_G^{sum}\). Then, for each \(m \in M\) and each \(I \subseteq \Gamma(m)\) the materialisation of \(P\) according to \((m, I)\) is defined as \(mat_m^I(S) = \langle U^I, P, \iota' \rangle\), with

\[
U^I = U \setminus (m(N_L) \setminus I) \cup N_L
\]

and for \(p \in C^2\) and \(q \in C^1 \setminus \{ sm \}\), letting \(\hat{m} = m \cup id_{U \setminus N_L}\):

\[
\iota'(q)(u) = \begin{cases} 1 & \text{if } u \in N_L \land (u, q, u) \in E_L \\ \iota(q)(\hat{m}(u)) & \text{else} \end{cases}
\]

\[
\iota'(p)(u_1, u_2) = \begin{cases} 1 & \text{if } u_1, u_2 \in N_L \land (u_1, p, u_2) \in E_L \\ \iota(p)(\hat{m}(u_1), \hat{m}(u_2)) & \text{else} \end{cases}
\]

\[
\iota'(sm)(u) = \begin{cases} 0 & \text{if } u \in N_L \\ \iota(sm)(\hat{m}(u)) & \text{else} \end{cases}
\]

The collection of all such logical structures is then defined as the materialisation of \(S\) with respect to \(P\):

\[
mat_P(S) = \{ S \mid \exists m : [\varphi_P]_m^S = 1 \} \quad \text{regular rule application}
\]

\[
\cup \{ mat_m^I(S) \mid [\varphi_P]_m^S = \frac{1}{2}, I \subseteq \Gamma(m) \} \quad \text{materialisations}
\]

The following theorem states that it is indeed sufficient to consider \(mat_P(S)\) instead of \(focus_P(S)\).
**Theorem 1.** Let $S$ be a 3-valued logical structure and $P$ a production rule. Then

\[ \text{mat}_P (S) \subseteq \text{focus}_P (S) \quad (3) \]

\[ \text{focus}_P (S) \sqsubseteq \text{mat}_P (S) \quad (4) \]

**Proof.** Since a detailed proof of this would be rather long, only the basic idea is presented here.

For (3) we observe that for any $S' \in \text{mat}_P (S)$ there exists by construction an assignment $m$ which satisfies $\varnothing_P S' = 1$. Using this assignment we can construct the function

\[
f : U^{S'} \to U^S \\
u \mapsto u \quad \text{if } u \in U^{S'} \cap U^S \\
n_l \mapsto m(n_l) \quad \text{if } n_l \in N_l\]

which turns out to be an embedding. Every $S' \in \text{mat}_P (S)$ thus meets the criteria of $\text{focus}_P (S) \Rightarrow (3)$.

Now, in order to show (4), let $S' \in \text{focus}_P (S)$. We need to show that there is an $S_m \in \text{mat}_P (S)$ such that $S' \sqsubseteq S_m$. We do this by performing a materialisation of $S$ using the information provided by $S'$. Let $m'$ be the assignment used to map $N_L$ into $U^{S'}$. We construct an assignment of $N_L$ into $U^{S'}$ by concatenating it with the embedding function $f : U^{S'} \to U^S$:

\[
m := f \circ m'
\]

Now we need to define the set $I$ of summary nodes we would like to keep in the materialisation. The only case where we would have to keep a summary node that is in the range of $m$, is if $h$ assigns more nodes in $S'$ to that summary node than would be strictly necessary to apply $P$. If we were to delete the summary node, the superfluous nodes could not be matched to any node in the materialisation. Thus, we construct the set $I$ by setting

\[
I = \{ u \in S \mid \exists u_1, u_2 \in U^{S'}, u_1 \in m'(N_L), u_2 \notin m'(N_L), f(u_1) = f(u_2) = u \}
\]

We can now set $S_m := \text{mat}_m (S)$ and obtain the embedding function $g$:

\[
g : U^{S'} \to U^{S_m} \\
u \mapsto m'^{-1}(u) \quad \text{if } u \in m'(N_L) \\
u \mapsto f(u) \quad \text{else}
\]

Thus we can construct for any $S' \in \text{focus}_P (S)$ a structure $S_m \in \text{mat}_P (S)$ such that $S' \sqsubseteq S_m \Rightarrow (4)$.

Figure 6 shows the result of applying materialisation on the starting shape graph using the $\text{EnterStation}$ production rule.
3.2 Coerce

After materialisation we apply the coerce operation defined in [SRW02] on the resulting shape graphs. Doing so serves two proposes: On the one hand we can identify inconsistencies in the shape graph (e.g. an empty track with a RailCab on it). On the other hand we can “sharpen” some predicate values of the shape graph in some cases. The latter can be found, for example, when looking at the materialised shape graphs of Fig. 6. There, the empty predicate has the value \( \frac{1}{2} \) for node \( t \). Yet, the RailCab \( r \) is on definitely on it. Hence, we can sharpen the predicate value of empty to 0.

To automatically detect such imprecisenesses we define the meaning of certain predicates by attaching meaning formulae to them. We do so especially for the instrumentation predicates, but also for empty and on (a RailCab should only be on at most one track at the same time) in our example. The meaning formula \( \alpha_p \) of a predicate \( p \) must be defined over the core-predicates \( C \). For empty we could define for example:

$$\alpha_{\text{empty}}(v) = T(v) \land \nexists r : (\text{RC}(r) \land \text{on}(r, v))$$

These meaning formulae can be used to derive the so-called compatibility constraints of an STS. The basic idea for this is to have constraints like \( \alpha_p (v) \Rightarrow p(v) \) and \( \neg \alpha_p (v) \Rightarrow \neg p(v) \). Compatibility constraints make, however, use of a new logical connective \( \triangleright \) replacing the \( \Rightarrow \) (see Table 2).

The idea of using the \( \triangleright \) connective is to have a stronger version of \( \Rightarrow \) such that constraints like e.g. \( \alpha_p (v) \Rightarrow p(v) \) are unfulfilled in some shape graph \( S \) if
\[ \begin{array}{c|ccc} \triangleright & 0 & \frac{1}{2} & 1 \\ \hline 0 & 1 & 1 & 1 \\ \frac{1}{2} & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{array} \]

Table 2: Truth table for \( \triangleright \)

\[ \alpha_p(v) |_S = 1 \] and \( p(v) |_S = \frac{1}{2} \), because, in that case, the predicate value of \( p \) could be sharpened to 1.

**Definition 17.** For some formula \( \varphi \) with free variables \( v_1, \ldots, v_k \) and some \( k \)-ary predicate \( p \in \mathcal{P} \), a compatibility constraint \( f \) is of one of the following forms:

\[
\begin{align*}
\varphi(v_1, v_2, \ldots, v_k) & \triangleright 0 \\
\varphi(v_1, v_2, \ldots, v_k) & \triangleright (v_1 = v_2) \\
\varphi(v_1, v_2, \ldots, v_k) & \triangleright \neg(v_1 = v_2) \\
\varphi(v_1, v_2, \ldots, v_k) & \triangleright p(v_1, v_2, \ldots, v_k) \\
\varphi(v_1, v_2, \ldots, v_k) & \triangleright \neg p(v_1, v_2, \ldots, v_k)
\end{align*}
\]

Let \( S \) be a shape graph. Let \( \Sigma \) be a set of compatibility constraints. We say \( S \) satisfies \( \Sigma \) (denoted by \( S \models \Sigma \)) if for every constraint \( f \) in \( \Sigma \) and every assignment \( Z \triangleright f |_S = 1 \) holds.

Please refer to [SRW02] for details about how to compute a set of compatibility constraints \( \Sigma \) from the meaning functions as described above. In our example the meaning formulae for \( \text{empty} \) would, for example, lead to the following compatibility constraints:

\[
\begin{align*}
T(v) \land \exists r : (RC(r) \land on(r, v)) & \triangleright \text{empty}(v) \\
\neg(T(v) \land \exists r : (RC(r) \land on(r, v))) & \triangleright \neg\text{empty}(v) \\
\text{empty}(v_2) & \triangleright \neg on(v_1, v_2)
\end{align*}
\]

Given a set of compatibility constraints, the \textit{coerce} operator can then be used to compute for a given shape graph \( S \) the most abstract shape graph \( S' \subseteq S \) such that \( S' \models \Sigma \). Note that there may be no shape graph \( S' \subseteq S \) such that \( S' \) satisfies the constraints. In that case \textit{coerce}(\( S \)) returns the nil value \( \bot \).

Figure 7 shows the result of the \textit{coerce} operation when applied on the materialised shape graphs of Fig. 6. Note how the \textit{empty} edge of the track \( t \) gets removed by \textit{coerce}. Also some other predicate values are sharpened. For example, the \( \frac{1}{2} \) valued edge from \( r \) to \( t' \) gets removed, because \( r \) is already definitely \textit{on} track \( t \).

Unfortunately, \textit{coerce}(\( S \)) \( \neq \bot \) for some shape graph \( S \) does not imply that there is a 2-valued logical structure \( G \) such that \( G \sqsubseteq S \) and \( G \models \Sigma \). For
example, consider a set of compatibility constraints that contains the constraints \(\neg \text{empty}(v) \triangleright \text{empty}(v)\) and \(\text{empty}(v) \triangleright \neg \text{empty}(v)\). Clearly, there is no 2-valued logical structure that could fulfill both constraints, but a shape graph can fulfill both by setting the predicate value of \text{empty} to \(\frac{1}{2}\) for all nodes. This leads to the following definition of well-formed compatibility constraints sets:

**Definition 18.** Let \(\Sigma\) be a set of compatibility constraint. We say \(\Sigma\) is a well-formed set of compatibility constraints, if for every 3-valued logical structure \(S\) with \(S \models \Sigma\), there also is a 2-valued \(G\) with \(G \cong S\) and \(G \models \Sigma\).

### 3.3 Production Application

After materialisation and coercion we can now directly apply the shape production rule with respect to some matching \(m\). Unlike a classical production rule a shape production rule also needs to specify how the instrumentation predicate values should get updated when the production is applied. For this purpose, we simply attach an update formula \(\gamma\) for the instrumentation predicates to the classical production rule and thus get a shape production rule.

**Definition 19.** A shape production rule \(P = ([L,R], \gamma)\) consists of a graph production rule \([L,R]\) and function \(\gamma\) mapping each instrumentation predicate \(p \in I\) and each node \(v \in N_R\) to a first-order predicate-update formula \(\varphi_{p,v}\) with free variables in \(N_L\).

The predicate-update formula \(\varphi_{p,v}\) specifies how the value of the instrumentation predicate \(p\) should be calculated for each \(v \in N_R\) of the new shape graph.
with respect to the predicate values of the old shape graph. For example, we could attach the following update formulae to the production rule EnterStation:

\[\varphi_{is\_colliding, r} = 0\]
\[\varphi_{is\_colliding, s} = 0\]
\[\varphi_{is\_colliding, t} = is\_colliding(t) \land \exists r_2, r_3 : ((r_2 \neq r) \land (r_3 \neq r) \land (r_3 \neq r_2) \land on(r_2, t) \land on(r_3, t))\]

Note that we make use of a free variable called \(r\) in the formula \(\varphi_{is\_colliding, t}\). When the production rule is applied to a shape graph \(S\), this free variable gets assigned to the individual in \(S\) that represents the \(r\) node of the left hand side of the production rule. The following definition formalises the shape production application.

**Definition 20.** Let \(P = (\langle L, R \rangle, \gamma)\) be a shape production rule and \(S = \langle U, \iota \rangle\) be a shape graph. The rule \(P\) can be applied to \(S\) if we find an injective function \(m : N_L \to U\) (again called a matching) such that for all \((n, p, n') \in E_L:\ i(p)(m(n), m(n')) = 1\) (or \(i(p)(m(n)) = 1\) for \(p \in C^1\)).

If \(m\) is a matching, then the application of \(P\) onto \(S\) with respect to the matching \(m\) is the structure \(S' = \langle U', \iota' \rangle\) with \(U' = (U \setminus m(N^-)) \cup N^+\) and \(\iota'\) defined as follows for \(p \in C^2, o \in C^1 \setminus \{sm\}, q \in I, m = m \cup \text{id}_{N^+}\), and \(u, u_1, u_2 \in U'\):

\[i'(o)(u) = \begin{cases} 0 & \text{if } (u, o, u) \in m(E_{del}) \\ 1 & \text{if } (u, o, u) \in \bar{m}(E_{new}) \\ i(o)(u) & \text{else} \end{cases}\]
\[i'(p)(u_1, u_2) = \begin{cases} 0 & \text{if } (u_1, p, u_2) \in m(E_{del}) \\ 1 & \text{if } (u_1, p, u_2) \in \bar{m}(E_{new}) \\ i(p)(u_1, u_2) & \text{else} \end{cases}\]
\[i'(sm)(u) = \begin{cases} 0 & \text{if } u \in N_{new} \\ i(sm)(u) & \text{else} \end{cases}\]
\[i'(q)(u) = \begin{cases} \llbracket\gamma(q, m^{-1}(u))\rrbracket^S_m & \text{if } u \in m(N_L) \\ \llbracket\gamma(q, u)\rrbracket^S_m & \text{if } u \in N_{new} \\ i(q)(u) & \text{else} \end{cases}\]

We write \(S \xrightarrow{P, m} S'\) if \(S'\) is the result of applying \(P\) with matching \(m\) to \(S\).

Figure 8 shows the result of applying EnterStation on the coerced shape graphs of Fig. 7.

Up until now, we did not make any restrictions on how the update formulae in \(\gamma\) should look like. Yet, we actually want the update formulae to "maintain a correct instrumentation" (definition is based on the corresponding property of update formulae in [SRW02]):
Definition 21. Let \( P = (\langle L, R \rangle, \gamma) \) be a shape production rule. Let \( \Sigma \) be a set of compatibility constraints. Let \( \Sigma_C \subseteq \Sigma \) be the set of compatibility constraints only related to core predicates. For some instrumentation predicate \( p \in \mathcal{I} \) let \( \alpha_p \), with free variable \( v \), denote the meaning function of \( p \). The shape production rule \( P \) maintains a correct instrumentation if for all \( p \in \mathcal{I} \) and for all 2-valued structures \( G = (U^G, \iota^G) \) on which \( G \models \Sigma \) holds and on which \( P \) can be applied such that \( G \xrightarrow{P,m} G' \) (for some matching \( m \)) with \( G' = (U^{G'}, \iota^{G'}) \) and \( G' \models \Sigma_C \), it holds for all \( u \in U^{G'} \):

\[
\iota^{G'}(p)(u) = \|\alpha_p\|^G_{m^{-1}(u)}
\]

The definition basically states that a production \( P \) maintains a correct instrumentation if for each valid embedded concrete graph \( G \) that leads to another graph \( G' \) when applying \( P \) which at least fulfills the compatibility constraints not related to instrumentation predicates, it holds that the evaluation of the meaning function \( \alpha_p \) in \( G' \) for each instrumentation predicate \( p \) matches the interpretation value of the instrumentation predicate \( p \) in \( G' \). Currently, we have to manually prove for each shape production rule that it maintains a correct instrumentation.

The following proposition shows that we can find a matching for a shape production \( P \) rule if \( \|\varphi_P\|_m = 1 \).

Proposition 1. Let \( S \) be a logical structure and \( P \) a shape production rule such that for an assignment \( m \) \( \|\varphi_P\|_m^S = 1 \) holds. Then the assignment \( m \) forms a matching for \( P \) in \( S \). If \( S \) was materialised from \( S' \) with respect to \( P \), then id forms a matching for \( P \) in \( S \).
4 Theoretical Results

In this section we describe how the methods introduced in Sect. 3 can be put together to form a verification algorithm. Afterwards, we analyse the correctness of the algorithm. That is, we show that the set of reachable graphs we compute with our shape-analysis based algorithm yields an over-approximation of the classical reachable graph set. Thus, if our algorithm does not find a forbidden pattern in the STS then a corresponding GTS will also never reach a forbidden pattern. If however a counter example is found, then we cannot be sure in every case that this counter example has an concrete counterpart in a GTS.

Using the methods described in Sect. 3 we can now, first of all, define the set of reachable shape graphs of a STS. To do so, we start by introducing a more convenient way to denote the shape graph transformation with respect to a production rule. Hence, we say that $S \xrightarrow{P} S'$ if $P$ is a shape production rule which can be applied on $S$ such that there is a $S_m \in \text{mat}_P(S)$ with $\text{coerce}(S_m) \xrightarrow{P} S'$. Shape graph transformation systems and its set of reachable shape graphs can then be defined as follows:

**Definition 22.** A shape graph transformation system $ST = (S_0, ((P_i, \gamma_i))_{i \in I})$ (that maintains a correct instrumentation) consists of an initial structure $S_0$ and a set of shape production rules $(P_i, \gamma_i), i \in I$ (that maintain a correct instrumentation). The set of reachable shape graphs of a shape graph transformation system $ST$ is defined inductively as follows:

$$\text{reach}(ST) = \max \left( \{S_0\} \cup \{\text{coerce}(S') \mid S \in \text{reach}(ST) \land i \in I \land S \xrightarrow{P_i} S'\} \right)$$

Here, $\max$ is defined for a set of shape graph $XS$ as in [SRW02] as:

$$\text{max}(XS) := XS \setminus \{X \mid \exists X' \in XS : X \sqsubseteq X' \land X' \not\sqsubseteq X\}$$

The definition of the set of reachable shape graphs already sketches how we compute it: We start with a set $XS$ containing only the starting shape graph and then apply for each shape graph in the the set $XS$ each production (using first materialisation, then coerce, then application). Finally we coerce the newly found shape graph, add it to the set $XS$ and apply the maximum operation on $XS$ afterwards. This is repeated as long as new shape graphs can be found (fixed point iteration).

This following theorem states that the algorithm is sound, i.e. we do not miss any of the reachable graphs:

**Theorem 2.** Let $\Sigma$ be a set of compatibility constraints. Let $S$ be a shape graph such that $S \models \Sigma$. Let $PS := ((P_i, \gamma_i))_{i \in I}$ be a set of shape production rules. It holds:

$$\bigcup_{G \subseteq S} \text{reach}((G, PS)) \subseteq \{G \mid G \text{ 2-valued } \land G \sqsubseteq S' \land S' \in \text{reach}((S, PS))\}$$
Lemma 1. Let $G$ be a 2-valued and $S$ a 3-valued logical structure such that $G \subseteq S$. Let $\Sigma$ be a set of compatibility constraints and $G \models \Sigma$, $S \models \Sigma$. Let $P = (\langle L, R \rangle, \gamma)$ be a shape production rule. If $G \xrightarrow{P} G'$ with $G' \models \Sigma$ then there exists a shape graph $S'$ such that $S \xrightarrow{\gamma} S'$, $G' \subseteq S'$, and $\text{coerce}(S') \neq \perp$.

Proof. Since $G \xrightarrow{P} G'$, there is some (injective) matching $m' : N_L \rightarrow U^G$ such that $P$ can applied to $G$ using the matching $m'$. Thus, $\llbracket \varphi_P \rrbracket^G_{m'} = 1$ holds. By this and because $G \subseteq S$ holds, we get $G \in \text{focus}(S)$. Hence, by Theorem 1 there must be a shape $S_m \in \text{mat}_P(S)$ with $G \subseteq S_m$. By Lemma 4 (see appendix) we get $\text{coerce}(S_m) \neq \perp$. Let $f : U^G \rightarrow U^{S_m}$ be the function that embeds $G$ in $S_m$ (also embeds $G$ in $\text{coerce}(S_m)$) as defined in Theorem 1. Let $S'$ be the shape we get, when applying $P$ to $\text{coerce}(S_m)$ using the matching $id$. We show that $\hat{f} := (f \cup N^+)_{|U^{G'}}$ embeds $G'$ in $S'$, i.e., $G' \subseteq \hat{f} S'$:

- $\hat{f}$ is surjective:
  Let $u \in U^{S'}$. If $u$ was already in $U^{S_m}$, then there is some $u' \in U^G$ such that $f(u') = u$. Obviously $u \notin N^-$, and since $f \circ m' = id$ (see construction of $f$ in Theorem 1), we also get $u' \notin m'(N^-)$ and thus $u' \in U^{G'}$.
  If $u$ was not already in $U^{S_m}$, then $u \in N^+$. In this case, we have $\hat{f}(u) = u$.

- $\langle \{ u \mid \hat{f}(u) = u' \} \rangle > 1 \subseteq \iota^{S'}(sm\langle u' \rangle)$:
  Let $u' \in U^{S'}$ such that there are two individuals $u_1, u_2 \in U^{G'}$ ($u_1 \neq u_2$) such that $\hat{f}(u_1) = u = \hat{f}(u_2)$. By construction of $\hat{f}$, we get that $u' \notin N^+$. Thus, $f(u_1) = u = f(u_2)$ holds and because $f$ is an embedding $\iota^{S'}(sm\langle u' \rangle) = \iota^{S_m}(sm\langle u' \rangle) = 1$ holds.

- $\iota^{G'}(p)(u_1, u_2, \ldots, u_k) \subseteq \iota^{S'}(p)(\hat{f}(u_1), \hat{f}(u_2), \ldots, \hat{f}(u_k))$:
  - $p \in C \setminus C^1$:
    If $(u_1, p, u_2) \in m'(E^-)$, then also $(\hat{f}(u_1), p, \hat{f}(u_2)) \in E^-$ (because $f \circ m' = id$). Thus:
    \[
    \iota^{G'}(p)(u_1, u_2) = 0 = \iota^{S'}(p)(u_1, u_2)
    \]
  Otherwise, we get:
    \[
    \iota^{G'}(p)(u_1, u_2) = \iota^{G}(p)(u_1, u_2) \subseteq \iota^{S_m}(p)(u_1, u_2)
    \]
  We show that $\iota^{S_m}(p)(u_1, u_2) = \iota^{S'}(p)(u_1, u_2)$ in this case by distinguishing some cases again. If $(\hat{f}(u_1), p, \hat{f}(u_2)) \in E^-$, we get $u_1, u_2 \in \text{ran}(m')$.

The correctness of the theorem is based on the following lemma:
by \( \iota^{S'}(sm)(u_1) = 0 = \iota^{S'}(sm)(u_2) \) and thus \((m'((\hat{f}(u_1)), p, m'((\hat{f}(u_2))) = (u_1, p, u_2) \in m'((E^-) \) (case covered above). If \((\hat{f}(u_1), p, \hat{f}(u_2)) \in E^+ \), we get analogical get \((u_1, p, u_2) \in \hat{m}'((E^+) \), which is also covered above. By construction of \( S' \), we can now conclude, that \( \iota^{S''}(p)(u_1, u_2) = \iota^{S'}(p)(u_1, u_2) \) holds in this “otherwise” case.

- \( p \in C^1 \setminus \{sm\} \):
  - For \( (u, p, u) \in \hat{m}'((E^-) \Leftrightarrow (\hat{f}(u), p, \hat{f}(u)) \in E^- \), it holds:
    \[
    \iota^{G'}(p)(u) = \iota^{S'}(p)(u)
    \]
  - For \( (u, p, u) \in \hat{m}'((E^+) \Leftrightarrow (\hat{f}(u), p, \hat{f}(u)) \in E^+ \) for \( \hat{m}' = m' \cup id_{N^+} \), it holds:
    \[
    \iota^{G'}(p)(u) = 1 = \iota^{S'}(p)(u)
    \]
  - Otherwise, we get:
    \[
    \iota^{G'}(p)(u) = \iota^{G}(p)(u) \subseteq \iota^{S_m}(p)(u) = \iota^{S'}(p)(u)
    \]

- \( p = sm \):
  - For \( u \in N^+ \Leftrightarrow \hat{f}(u) \in N^+ \), it holds:
    \[
    \iota^{G'}(sm)(u) = 0 = \iota^{S'}(sm)(u) = \iota^{S'}(sm)(\hat{f}(u))
    \]
  - Otherwise, we get:
    \[
    \iota^{G'}(sm)(u) = \iota^{G}(sm)(u) \subseteq \iota^{S_m}(sm)(u) = \iota^{S'}(sm)(u)
    \]

- \( p \in I \):
  - For \( u \in ran m' \Leftrightarrow \hat{f}(u) \in N_L \), it holds:
    \[
    \iota^{G'}(p)(u) = \begin{bmatrix} \gamma(p, m^{-1}(u)) \end{bmatrix} \begin{bmatrix} G_{m'} \end{bmatrix} \\
    = \begin{bmatrix} \gamma(p, \hat{f}(u)) \end{bmatrix} \begin{bmatrix} G_{m'} \end{bmatrix} \\
    \leq \begin{bmatrix} \gamma(p, \hat{f}(u)) \end{bmatrix} \begin{bmatrix} S_m \end{bmatrix} \\
    = \begin{bmatrix} \gamma(p, \hat{f}(u)) \end{bmatrix} \begin{bmatrix} S_m \end{bmatrix} \\
    = \iota^{S'}(p)(\hat{f}(u))
    \]
  - For \( u \in N^+ \Leftrightarrow \hat{f}(u) \in N^+ \), it holds:
    \[
    \iota^{G'}(p)(u) = \begin{bmatrix} \gamma(p, u) \end{bmatrix} \begin{bmatrix} G_m \end{bmatrix} \\
    = \begin{bmatrix} \gamma(p, \hat{f}(u)) \end{bmatrix} \begin{bmatrix} G_m \end{bmatrix} \\
    \leq \begin{bmatrix} \gamma(p, \hat{f}(u)) \end{bmatrix} \begin{bmatrix} S_m \end{bmatrix} \\
    = \begin{bmatrix} \gamma(p, \hat{f}(u)) \end{bmatrix} \begin{bmatrix} S_m \end{bmatrix} \\
    = \iota^{S'}(p)(\hat{f}(u))
    \]

\[\text{Lemma 3}\]
Otherwise, we get:
\[
\iota^{G'}(p)(u) = \iota^{G}(p)(u) \sqsubset \iota^{S_m}(p)(u) = \iota^{S}(p)(u)
\]
By \(G' \sqsubseteq S'\) and \(G' \models \Sigma\), \(\text{coerce}(S') \neq \bot\) finally follows by Lemma 4.

We can now prove Theorem 2.

**Proof of Theorem 2.** Let \(G' \in \text{reach}((G, PS))\) for some 2-valued \(G \sqsubseteq G'\). We need to show that there is a \(S' \in \text{reach}((S, PS))\) such that \(G' \sqsubseteq S'\). By definition of \(\text{reach}\) there is a chain of production rules \(\Pi = (P_1, \gamma_1), (P_2, \gamma_2), \ldots, (P_k, \gamma_k)\), such that \(G \xrightarrow{\Pi} G''\) and \(\text{coerce}(G'') = G'' = G'\). We prove the lemma by induction over the length \(k\) of \(\Pi\):

I.B. For \(k = 0\), the claim is trivially true.

I.H. For every \(G' \in \text{reach}((G, PS))\) reachable by a production of length \(k - 1\), there is \(S' \in \text{reach}((S, PS))\) such that \(G' \sqsubseteq S'\).

I.S. \(((k - 1) \to k)\) Let \(\Pi' := (P_1, \gamma_1), (P_2, \gamma_2), \ldots, (P_{k-1}, \gamma_{k-1})\). Let \(G_{k-1} = (G_{k-1} \models \Sigma)\) be a two-valued structures such that \(G_0 \xrightarrow{\Pi'} G_{k-1}\), \(\text{coerce}(G_{k-1}) \xrightarrow{(P_k, \gamma_k)} G''\) and \(\text{coerce}(G'') = G'' = G'\). By I.H. there is a \(S_{k-1} \in \text{reach}((S, PS))\) such that \(G_{k-1} = \text{coerce}(G_{k-1}) \sqsubseteq S_{k-1}\). By definition of \(\text{reach}\) it also holds \(S_{k-1} \models \Sigma\). We can thus apply Lemma 5 (see Appendix) and obtain a structure \(S''\) such that \(S_{k-1} \xrightarrow{(P_k, \gamma_k)} S''\), \(G' \sqsubseteq S''\), and \(\text{coerce}(S'') \neq \bot\). Let \(S' := \text{coerce}(S'')\). If \(S' \in \text{reach}((S, PS))\) we are done. Otherwise by definition of \(\text{max}\), there is a structure \(\tilde{S}\) such that \(S' \sqsubseteq \tilde{S}\), and thus \(G' \sqsubseteq \tilde{S}\).

The inversion of the lemma above is not true in general. That is why we only yield an over-approximation of the classical set of reachable graphs. Nevertheless, in the special case that there are no instrumentation predicates and no compatibility constraints, the inversion can actually be shown:

**Proposition 2.** Assume \(\mathcal{I} = \emptyset\). Let \(\Sigma = \emptyset\). Let \(S\) be a shape graph. Let \(P = ((L, R), \gamma)\) be a shape production rule. If \(S \xrightarrow{P} S'\) then for every 2-valued \(G' \sqsubseteq S'\) with \(G' \models \Sigma\) there exists a 2-valued logical structure \(G\) such that \(G \xrightarrow{P} G'\), \(G \sqsubseteq S\), and \(G \models \Sigma\).

**Proof.** Let \(S_m \in \text{mat}_P(S)\) such that \(\text{coerce}(S_m) \xrightarrow{P, \text{id}, \gamma} S'\). Let \(S_c\) denote \(\text{coerce}(S_m)\). Let \(f : U^{G'} \to U^S\) be the function that embeds \(G'\) in \(S'\). We define a tree-valued structure \(S_{\text{aux}} = (U^{S_{\text{aux}}}, I^{S_{\text{aux}}})\) and construct \(G\) such that \(G \sqsubseteq S_{\text{aux}}\) and \(U^G = U^{S_{\text{aux}}}\). We set \(U^{S_{\text{aux}}} = U^{G'} \setminus f^{-1}(N^+) \cup N^-\) (i.e., undo the addition of the individuals in \(N^+\) by removing them and undo the removal of the individuals in \(N^-\) by adding them again). We define \(\tilde{f} : U^G \to U^{S'}\) as \(\tilde{f} := (f \cup \text{id}_{N^-})|_{U^G}\).
For $o \in C^1 \setminus \{sm\}$, $p \in C \setminus C^1$, $q \in \mathcal{I}$, we define $\iota^{S_{aux}}$ as follows:

$$
\iota^{S_{aux}}(o)(u) = \begin{cases} 
1 & \text{if } (\hat{f}(u), p, \hat{f}(u)) \in E_L \\
\iota^{S_\epsilon}(o)(\hat{f}(u)) & \text{if } u \in N^- \\
\iota^{G'}(o)(u) & \text{else}
\end{cases}
$$

$$
\iota^{S_{aux}}(p)(u_1, u_2) = \begin{cases} 
1 & \text{if } (\hat{f}(u_1), p, \hat{f}(u_2)) \in E_L \\
\iota^{S_t}(p)(\hat{f}(u_1), \hat{f}(u_2)) & \text{if } u_1 \in N^- \lor u_2 \in N^- \\
\iota^{G'}(p)(\hat{f}(u_1), \hat{f}(u_2)) & \text{if } (f(u_1), p, f(u_2)) \in E^+ \\
\iota^{G'}(p)(u_1, u_2) & \text{else}
\end{cases}
$$

$$
\iota^{S_{aux}}(sm)(u) = 0
$$

We need to show that $S_{aux}$ fulfills $\Sigma$. However, this is trivially true, because we assumed $\Sigma = \emptyset$. Since $\Sigma$ is a well-formed set of compatibility constraints, there must be a two-valued structure $G = (U^G, \iota^G)$ such that $G \subseteq S_{aux}$ and $G \models \Sigma$. Let w.l.o.g. be $U^G = U^{S_{aux}}$ (this is sound, because $\iota^{S_{aux}}(sm)(u) = 0$ for all $u \in U^{S_{aux}}$). We need to show that for this $G$, it holds (for $m : N_L \to U^G$ with $m(u) := \hat{f}^{-1}(u)$ for every $u \in N_L$):

$$
\begin{align*}
G & \subseteq \overline{f} \cdot S_c \\
G & \models \overline{m} \cdot \overline{G}'
\end{align*}
$$

We start by showing that $\hat{f}$ is indeed an embedding. Note that by $S_c \subseteq S_m \subseteq S$, we then also get $G \subseteq S$.

- $\hat{f}$ is surjective:
  First of all, note, that $\hat{f}(u) \in U^{S'}$ for all $u \in U^G \setminus N^-$, but since for all $v \in U^G$ it holds $v \not\in f^{-1}(N^+)$, we also have $\hat{f}(v) \in U^{S_c}$. Thus, $\hat{f} : U^G \to U^{S_c}$ is well-defined. Now, let $u \in U^{S_c}$. If $u \in N^-$ then we have $\hat{f}(u) = u$. Otherwise, we have $u \in U^{S'}$ and by this there is a preimage $v \in U^{G'}$ such that $f(v) = u$. Clearly, for this $v$ it holds $v \not\in f^{-1}(N^+)$ and thus $v \in U^G$.

- $\{ \{ u \mid \hat{f}(u) = u' \} \mid > 1 \} \subseteq \iota^S(sm)(u')$:
  Let $u_1, u_2 \in U^G$ with $u_1 \neq u_2$ such that for some $u' \in U^G$ it holds $\hat{f}(u_1) = u' = \hat{f}(u_2)$. By construction of $\hat{f}$, we get $u' \not\in N^-$, $u_1 \not\in N^-$, and $u_2 \not\in N^-$. Thus, $u_1, u_2 \in U^{G'}$. Since $f$ is an embedding, it follows:

  $$(\{ \{ u \mid \hat{f}(u) = u' \} \mid > 1 \} \subseteq \iota^S(sm)(u') = \iota^S(sm)(u')$$

- $\iota^G(p)(u_1, u_2, \ldots, u_k) \subseteq \iota^{S_c}(\hat{f}(u_1), \hat{f}(u_2), \ldots, \hat{f}(u_k))$:
\( p \in C \setminus C^1:\) For \((\hat{f}(u_1), p, \hat{f}(u_2)) \in \mathcal{E}_L\), we have:

\[
\iota^G(p)(u_1, u_2) = \iota^{S_{\text{aux}}}(p)(u_1, u_2) = 1 \sqsubseteq 1 \quad \text{if } \exists j : \varphi_{\text{aux}}^j \subseteq \iota_{S_{\text{aux}}}(p)(\hat{f}(u_1), \hat{f}(u_2))
\]

For \(u_1 \in \mathcal{N}^- \vee u_2 \in \mathcal{N}^- \) or \((\hat{f}(u_1), p, \hat{f}(u_2)) \in \mathcal{E}^+, \) we get:

\[
\iota^G(p)(u_1, u_2) \sqsubseteq \iota^{S_{\text{aux}}}(p)(u_1, u_2) = \iota_{S_{\text{aux}}}(p)(\hat{f}(u_1), \hat{f}(u_2))
\]

Otherwise, we get:

\[
\iota^G(p)(u_1, u_2) = \iota^{S_{\text{aux}}}(p)(u_1, u_2) = \iota^G(p)(u_1, u_2) \sqsubseteq \iota^{S'}(p)(\hat{f}(u_1), \hat{f}(u_2)) = \iota^{S'}(p)(\hat{f}(u_1), \hat{f}(u_2))
\]

By definition of production application, we get

\[
\iota^{S'}(p)(\hat{f}(u_1), \hat{f}(u_2)) \neq \iota^G(p)(\hat{f}(u_1), \hat{f}(u_2))
\]

only if \((\hat{f}(u_1), p, \hat{f}(u_2)) \in \mathcal{E}^- \cup \mathcal{E}^+.\) Both cases are considered explicitly above \((\hat{f}(u_1), p, \hat{f}(u_2)) \in \mathcal{E}^- \Rightarrow (\hat{f}(u_1), p, \hat{f}(u_2)) \in \mathcal{E}_L.)\) Thus, in this case, we get:

\[
\iota^G(p)(u_1, u_2) \sqsubseteq \iota^{S'}(p)(\hat{f}(u_1), \hat{f}(u_2)) = \iota^G(p)(\hat{f}(u_1), \hat{f}(u_2))
\]

• \( p \in C^1 \setminus \{\text{sm}\}::\) For \((\hat{f}(u), p, \hat{f}(u)) \in \mathcal{E}_L,\) we have:

\[
\iota^G(p)(u) = \iota^{S_{\text{aux}}}(p)(u) = 1 \sqsubseteq 1 \quad \text{if } \exists j : \varphi_{\text{aux}}^j \subseteq \iota_{S_{\text{aux}}}(p)(\hat{f}(u))
\]

For \(u \in \mathcal{N}^- \) or \((\hat{f}(u), p, \hat{f}(u)) \in \mathcal{E}^+, \) we get:

\[
\iota^G(p)(u) \sqsubseteq \iota^{S_{\text{aux}}}(p)(u) = \iota_{S_{\text{aux}}}(p)(\hat{f}(u))
\]

Otherwise, we get:

\[
\iota^G(p)(u) = \iota^{S_{\text{aux}}}(p)(u) = \iota^G(p)(u) \sqsubseteq \iota^{S'}(p)(\hat{f}(u)) = \iota^{S'}(p)(\hat{f}(u))
\]

By definition of production application, we get

\[
\iota^{S'}(p)(\hat{f}(u)) \neq \iota^G(p)(\hat{f}(u))
\]

only if \((\hat{f}(u), p, \hat{f}(u)) \in \mathcal{E}^- \cup \mathcal{E}^+.\) Both cases are considered explicitly above \((\hat{f}(u), p, \hat{f}(u)) \in \mathcal{E}^- \Rightarrow (\hat{f}(u), p, \hat{f}(u)) \in \mathcal{E}_L.)\) Thus, in this case, we get:

\[
\iota^G(p)(u) \sqsubseteq \iota^{S'}(p)(\hat{f}(u)) = \iota^G(p)(\hat{f}(u))
\]
\* \( p = \text{sm} \):

Since \( G \) is two-valued, we get:

\[
\iota^G(\text{sm})(u) = 0 \subseteq \iota^S(\text{sm})(\hat{f}(u))
\]

Thus, we have shown, that \( G \subseteq \gamma S \subseteq \) holds. We are left to show that \( G \xrightarrow{\tilde{F}, m} G' \) holds. To to so, we, first of all, need to show that \( P \) can be applied to \( G \) when using the matching \( m \). Thus, let \((u_1, p, u_2) \in E_L \). Let w.l.o.g. \( p \) be binary. We need to show that \( \iota^G(p)(m(u_1), m(u_2)) = 1 \). By construction, for this property its sufficient to show that \((\hat{f}(m(u_1)), p, \hat{f}(m(u_2))) \in E_L \). By construction \( m : N_L \to U^G \) is defined as \( m(u) = \hat{f}^{-1}(u) \) for \( u \in N_L \) (note that \( m \) is well-defined, because \( \hat{f} \) is a embedding function and \( \iota^S(\text{sm})(v) = 0 \) for \( v \in N_L \)). Thus:

\[
(\hat{f}(m(u_1)), p, \hat{f}(m(u_2))) = (\hat{f}(\hat{f}^{-1}(u_1)), p, \hat{f}(\hat{f}^{-1}(u_2))) = (u_1, p, u_2) \in E_L
\]

Hence, we can apply \( P \) to \( G \) using the matching \( m \). Let \( G'' = (U^{G''}, \iota^{G''}) \) be the result of this operation. We need to show that \( G'' \simeq G' \) holds. To simplify the notions of the following proof, assume w.l.o.g. that \( \hat{f}^{-1}(N^+) = N^+ \). In this case, we can simply show \( G'' = G' \) (without the need of an isomorphism between \( G'' \) and \( G' \)). It holds:

\[
U^{G''} = (U^G \setminus m(N^-)) \cup N^+
= (U^G \setminus \hat{f}^{-1}(N^-)) \cup N^+
= ((U^G \setminus f^{-1}(N^+) \cup N^-) \setminus \hat{f}^{-1}(N^-)) \cup N^+
= U^G \setminus f^{-1}(N^+) \cup (N^- \setminus \hat{f}^{-1}(N^-)) \cup N^+
= U^G \setminus f^{-1}(N^+) \cup (N^- \setminus N^-) \cup N^+
= U^G \setminus f^{-1}(N^+) \cup N^+
= U^{G'}
\]

Let \( \hat{m} := m \cup id_{N^+} \). To show the equality of \( \iota^{G''} \) and \( \iota^{G'} \) we distinguish some cases:

- \( o \in C^1 \setminus \{\text{sm}\} \):
  
  For \( u \in U^{G''} \) and \((u, o, u) \in m(E^-) \Leftrightarrow (u, o, u) \in \hat{f}^{-1}(E^-) \Leftrightarrow (\hat{f}(u), o, \hat{f}(u)) \in E^- \), it holds:

  \[
  \iota^{G''}(o)(u) = 0 = \iota^{S'}(o)(\hat{f}(u)) = \iota^{S'}(o)(f(u))
  \]

By \( \iota^{G'}(o)(u) \subseteq \iota^{S'}(o)(f(u)) \), we also get \( \iota^{G'}(o)(u) = 0 \). For \( u \in U^{G''} \) and \((u, o, u) \in \hat{m}(E^+) \Leftrightarrow (u, o, u) \in \hat{f}^{-1}(E^+) \Leftrightarrow (\hat{f}(u), o, \hat{f}(u)) \in E^+ \), it holds:

\[
\iota^{G''}(o)(u) = 1 = \iota^{S'}(o)(\hat{f}(u)) = \iota^{S'}(o)(f(u))
\]
By \( i^{G'}(o)(u) \subseteq i^{S'}(o)(f(u)) \), we also get \( i^{G'}(o)(u) = 1 \). Otherwise, it holds:

\[
i^{G''}(o)(u) = i^{G}(o)(u) = i^{S_{\text{out}}}(o)(u)
\]

If \((f(u), o, f(u)) \notin E_L\), we get \( i^{S_{\text{out}}}(o)(u) = i^{G'}(o)(u) \) and are done. If \((f(u), o, f(u)) \in E_L\), it holds:

\[
i^{S_{\text{out}}}(o)(u) = 1 \\Leftrightarrow i^{S}(o)(f(u)) = i^{S'}(o)(f(u))
\]

By \( i^{G'}(o)(u) \subseteq i^{S'}(o)(f(u)) \), we then also get \( i^{G'}(o)(u) = 1 \).

\(-\ p \in C \setminus C'\):

For \( u_1, u_2 \in U^{G''} \) and \((u_1, o, u_2) \in m(E^-) \Leftrightarrow (u_1, o, u_2) \in \hat{f}^{-1}(E^-) \Leftrightarrow (\hat{f}(u_1), o, \hat{f}(u_2)) \in E^-\), it holds:

\[
i^{G''}(o)(u_1, u_2) = 0 \iff i^{S'}(o)(\hat{f}(u_1), \hat{f}(u_2)) = i^{S}(o)(f(u_1), f(u_2))
\]

By \( i^{G'}(o)(u_1, u_2) \subseteq i^{S'}(o)(f(u_1), f(u_2)) \), we also get \( i^{G'}(o)(u_1, u_2) = 0 \). For \( u_1, u_2 \in U^{G''} \) and \((u_1, o, u_2) \in \hat{m}(E^+) \Leftrightarrow (u_1, o, u_2) \in \hat{f}^{-1}(E^+) \Leftrightarrow (\hat{f}(u_1), o, \hat{f}(u_2)) \in E^+\), it holds:

\[
i^{G''}(o)(u_1, u_2) = 1 \iff i^{S'}(o)(\hat{f}(u_1), \hat{f}(u_2)) = i^{S}(o)(f(u_1), f(u_2))
\]

By \( i^{G'}(o)(u_1, u_2) \subseteq i^{S'}(o)(f(u_1), f(u_2)) \), we also get \( i^{G'}(o)(u_1, u_2) = 1 \). Otherwise, it holds:

\[
i^{G''}(o)(u_1, u_2) = i^{G}(o)(u_1, u_2) = i^{S_{\text{out}}}(o)(u_1, u_2)
\]

If \((f(u_1), o, f(u_2)) \notin E_L\), we get \( i^{S_{\text{out}}}(o)(u_1, u_2) = i^{G'}(o)(u_1, u_2) \) and are done. If \((f(u_1), o, f(u_2)) \in E_L\), it holds:

\[
i^{S_{\text{out}}}(o)(u_1, u_2) = 1 \iff i^{S}(o)(f(u_1), f(u_2)) = i^{S'}(o)(f(u_1), f(u_2))
\]

By \( i^{G'}(o)(u_1, u_2) \subseteq i^{S'}(o)(f(u_1), f(u_2)) \), we then also get \( i^{G'}(o)(u_1, u_2) = 1 \).

\(-\ sm\):

For every \( u \in U^{G''} \) it holds:

\[
i^{G''}(sm)(u) = 0 = i^{G'}(sm)(u)
\]

\( \Box \)

In the special case \( I = \emptyset \) and \( \Sigma = \emptyset \) we thus get in Theorem 2 the equality of the set of reachable graphs instead of the subset relation (over-approximation). Figure 9 shows, however, that Proposition 2 does not hold in general if there are compatibility constraints and instrumentation predicates. In Fig. 9 the compatibility constraints enforce for any 2-valued graph \( G \subseteq \Sigma \), that at most one of the
G is not valid.
\((G \models \Sigma \) does not hold\)

is-colliding should be 1, but in that case, \(G\) would not be embedded in \(S\) anymore

Fig. 9: Counterexample for a generalised version of Proposition 2
For the sake of completeness, we also need to relate the set of reachable graphs of a GTS that is represented using a classical approach with the set of reachable graphs of a GTS that is represented as STS (Theorem 2 assumes a representation as STS). The following lemma shows the direct relationship between such shape graph transformation systems and graph transformation systems. It also indicates why we are interested in shape productions that maintain a correct instrumentation.

**Lemma 2.** Let \( \Sigma \) be a set of compatibility constraints. Let \( \Sigma_C \subseteq \Sigma \) the set of compatibility constraints only related to core predicates. Let \( G_0 = (N, E) \) be a graph with \( ls(G_0) \models \Sigma_C \). We extend the \( ls(\cdot) \) operator to \( ls_T(\cdot) \), which canonically takes instrumentation predicates into account. We define \( ls_T(G) := (\{ \iota^{ls}(G), \iota^{ls}(G) \cup \iota' \}, \iota' \) where \( \iota' \) is defined for \( q \in I \) with meaning function \( \alpha_q \) (free(\( \alpha_q \)) = \{ v \}) and \( u \in U^{ls(G)} \) as follows:

\[
i'(q)(u) = [\alpha_q]^{ls(G)}_{[v \mapsto u]}
\]

Let \( GT = (G_0, (P_i)_{i \in I}) \) be a graph transformation system. Assume, that we can only apply a production \( P \in (P_i)_{i \in I} \) to a graph \( G \) if for the resulting graph \( G' \) it holds \( ls(G') \models \Sigma_C \). Let \( ST = (ls_T(G_0), ((P_i, \gamma_i))_{i \in I}) \) a shape graph transformation system that maintains a correct instrumentation. It holds:

\[
ls_T(\text{reach}(GT)) = \text{reach}(ST)
\]

**Proof.** “\( \subseteq \)”: Let \( G' \in \text{reach}(GT) \). We need to show that \( ls_T(G') \in \text{reach}(ST) \).

Let be \( \Pi = (P_1, P_2, \ldots, P_k) \) be a chain of (concrete) productions such that:

\[
G_0 \xrightarrow{P_1} G_1 \xrightarrow{P_2} \cdots \xrightarrow{P_k} G_k = G'
\]

By Proposition 4 we know that there is \( G'_1 \) such that \( ls_T(G_0) \xrightarrow{(P_1, \gamma_1)} G'_1 \) and \( G'_1 = ls(G_1) \) except for the instrumentation predicates. The value of \( \iota^{G'_1}(q)(u) \), however, must be \( [\alpha_q]^{G'_1}_{[v \mapsto u]} \), because the shape production remains a correct instrumentation and \( ls(G_1) \models \Sigma_C \). Thus, we get \( G'_1 = ls_T(G_1) \) and \( G'_1 \models \Sigma \). Inductively, it follows:

\[
ls_T(G_0) \xrightarrow{(P_1, \gamma_1)} ls_T(G_1) \xrightarrow{(P_2, \gamma_2)} \cdots \xrightarrow{(P_k, \gamma_k)} ls_T(G')
\]

Furthermore, since \( coerce \) applied to a two-valued structure can only result either in \( \perp \) or the structure itself, it also holds:

\[
ls_T(G_0) \xrightarrow{\Pi_T} ls_T(G')
\]

Finally, since \( max \) has no effect on sets of two-valued structures (a two-valued structure can only be embedded in a other two-valued structure if they are isomorphic), we get \( ls_T(G') \in \text{reach}(ST) \).
\[ \overset{\epsilon}{\supseteq} \]: Let \( G'' \in \text{reach}(ST) \). There is a chain of productions

\[ \Pi := (P_1, \gamma_1), (P_2, \gamma_2), \ldots, (P_k, \gamma_k) \]

such that

\[ \text{ls}_{\Sigma}(G_0) \xrightarrow{(P_1, \gamma_1)} G_1' \xrightarrow{(P_2, \gamma_2)} \ldots \xrightarrow{(P_k, \gamma_k)} G_k' = G'' \]

By Proposition 4, we can also apply the production \( P_1 \) (using the same matching) to \( G_0 \) and get \( G_1 \). We also know that \( G_1' = \text{ls}(G_1) \) except for the instrumentation predicates. The value of \( \iota_{G_1'}(q) \) for \( q \in \mathcal{I} \) can, however, only be \( \|\alpha_{q} \|_{\nu_{1} \to \nu_{1}} \) since \( G_1' \models \Sigma \). Thus \( G_1' = \text{ls}_{\Sigma}(G_1) \). Inductively, it follows

\[ G_0 \xrightarrow{P_1} G_1 \xrightarrow{P_2} \ldots \xrightarrow{P_k} G_k =: G' \]

where \( G_1' = \text{ls}_{\Sigma}(G_1), \ldots, G_k' = \text{ls}_{\Sigma}(G_k) \). Thus, \( G'' = \text{ls}_{\Sigma}(G) \) and \( G' \in \text{reach}(GT) \). \( \square \)

The lemma shows that it is indeed sound to use graphs and 2-valued logical structures synonymously even with respect to production application.

The following definitions introduces the notion of a shape graph containing a forbidden pattern:

**Definition 23.** Let \( \Sigma \) be a set of compatibility constraints. Let \( F \) be a 2-valued, \( G \) a 2-valued, and \( S \) a 3-valued logical structure. We say \( G \) contains the forbidden pattern \( F \) iff there is an injective morphism \( m : F \to G \). We say \( S \) contains the forbidden pattern \( F \) iff there is a 2-valued \( G' \subseteq S \) such that \( G' \) contains the forbidden pattern \( F \) and \( G' \models \Sigma \).

Using this definition we can finally formulate the property of our algorithm that if we do not find a shape graph containing a forbidden pattern, then there is also no reachable concrete graph that contains some forbidden pattern (i.e., the system is safe).

**Theorem 3.** Let \( \Sigma \) be a set of compatibility constraints. Let \( F \) be a set of 2-valued logical structures (forbidden patterns). Let \( G \) be a 2-valued and \( S \) be a 3-valued logical structure such that \( G \subseteq S \). Let \( PS := ((P_i, \gamma_i))_{i \in I} \) be a set of shape production rules. If there is a 2-valued logical structure \( G' \in \text{reach}((G, PS)) \) containing the forbidden pattern \( F \subseteq G \), then there also a shape graph \( S' \in \text{reach}((S, PS)) \) containing the forbidden pattern \( F \).

**Proof.** As shown in Theorem 2 there is structure \( S' \in \text{reach}((S, PS)) \) such that \( G' \subseteq S' \). Because \( G' \) contains \( F \), \( S' \) also contains \( F \). \( \square \)

There is still one piece missing to complete our verification algorithm: We need a method to algorithmically check if some shape graph \( S \) contains some
forbidden pattern $F$. The following proposition introduces such a method. The idea is to interpret the forbidden pattern as the left side of a production rule and then simply apply materialisation with respect to this production on $S$. If $S$ really contains $F$ then one of the materialised shape graphs should be valid ($\text{coerce}$ does not yield $\bot$). The other direction only holds if $\Sigma$ is well-formed (otherwise there still might be no concrete graph embedded in the valid materialised shape graph).

**Proposition 3.** Let $\Sigma$ be a set of compatibility constraints. Let $F$ be a 2-valued and $S$ a 3-valued logical structure with $S \models \Sigma$. Let $P = ((F, \bot), \text{id})$ be a shape production rule. If $S$ contains the forbidden pattern $F$ then $\text{coerce}(\text{mat}_P(S)) \neq \emptyset$ holds. If $\Sigma$ is well-formed then $S$ contains the forbidden pattern $F$ if and only if $\text{coerce}(\text{mat}_P(S)) \neq \emptyset$ holds.

**Proof.** We have to show two directions:

"$\Rightarrow$" If $S$ contains $F$ there is by definition a two-valued $G' \subseteq S$ such that $G'$ contains $F$ and $G' \models \Sigma$. For this $G'$, there is an injective morphism $m : F \rightarrow l^{-1}(G)$. Thus, $[\varphi_P]_{m}^{G'} = 1$ holds and we get $G' \in \text{focus}_P(S)$. By Theorem 1 we know that there is a structure $S'$ such that $S' \in \text{mat}_P(S)$ with $G' \subseteq S'$. By Lemma 4 we also get that $\text{coerce}(S') \neq \bot$ holds. Therefore, $\text{coerce}(\text{mat}_P(S)) \neq \emptyset$ holds.

"$\Leftarrow$" Let $S' \in \text{mat}_P(S)$ such that $\text{coerce}(S') \neq \bot$. By definition of $\text{mat}_P$, there is an assignment $m$ such that $[\varphi_P]_{m}^{S'} = 1$ holds. Also, by Theorem 1 it holds: $S' \subseteq S$. Since $\Sigma$ is well-formed, there is a two-valued $G'$ such that $G' \subseteq \text{coerce}(S')$ and $G' \models \Sigma$. Because of Lemma 3 there must be an assignment $m'$ such that $[\varphi_P]_{m'}^{G'} = 1$ holds. Because $G' \subseteq \text{coerce}(S') \subseteq S' \subseteq S$, $G'$ contains $F$ and hence $S$ also contains $F$. $\square$

This completes our algorithm. Note that our algorithm may not terminate, if $\text{reach}((S, PS))$ is infinite in size and $F$ is not contained in any of these graphs. The question arises, if this fact is due to a poor design of our algorithm or if it is not possible to design a algorithm that decides if $(S, PS)$ eventually contains a forbidden pattern $F \in F$. The following theorem shows that the latter is true.

**Theorem 4.** The problem to check whether there is a $S' \in \text{reach}((S, PS))$ that contains a forbidden pattern $F \in F$ for some shape graph $S$, a set of shape production rules $PS$ and a set of 2-valued logical structures $F$, is undecidable in general.

**Proof.** Assume there is a algorithm $A$ that can decide (in finite time) whether there is a $S' \in \text{reach}((S, PS))$ that contains a $F \in F$ for every given $S$, $PS$. We show that we use this algorithm $A$ to decide whether a given turning machine holds or not (i.e., solving the halting problem). To this end we model a (reachable) configuration of a given turning machine as a graph $S' \in \text{reach}((S, PS))$
and the initial configuration as the starting graph $S$. The transitions of the state machine of the turing machine are modelled as production rules. The set $\mathcal{F}$ models the set of accepting states. Because we cannot explicitly model the infinite tape of e.g. the initial configuration, we use the fact that we can add individuals using shape productions. Thus we can have a $endTape$ labelled individual at a given configuration addition production that takes the $endTape$ as left side and appends a blank labelled individual to the left (or right) of the $endTape$. Formally, let $M = (Q, \Gamma, b, \Sigma, \delta, q_0, FS)$ be a turing machine. Let w.l.o.g. be $endTape, pointsTo, right, left, sm \not\in \Gamma$. We define the set of predicates as follows:

$$\mathcal{P} := \Gamma \cup \{endTape, pointsTo, right, left, sm\} \cup Q$$

All the predicates are unary core predicates except for $right, left,$ and $pointsTo,$ which are binary core predicates (i.e. $C^1 = \Gamma \cup \{endTape, sm\} \cup Q, \mathcal{C} \setminus \mathcal{C}^1 = \{pointsTo, right, left\}, \mathcal{I} = \emptyset$). We define the start shape $S$ as in Figure 10. Let $P_{extendLeft}$ and $P_{extendRight}$ shape productions as defined by Figure 11 and Figure 12 respectively. Let $P_{(q,s,q',s',d)}$ and $P_{(q,s,q',s',d)}$ for $q, q' \in Q, s, s' \in \Gamma$ be shape productions as defined by Figure 13 and Figure 14 respectively. We define:

$$PS := \{P_{extendLeft}, P_{extendRight}\} \cup \{P_{(q,s,q',s',d)} \mid (q, s, q', s', d) \in \delta\}$$

Finally for some accepting state $f \in FS$, we define $V_f$ as the shape defined by Figure 15. We then finally define $\mathcal{F}$ as follows:

$$\mathcal{F} := \{V_f \mid f \in FS\}$$

It holds:

$$M \text{ halts on input } \epsilon \iff \exists S' \in \text{reach}(S, PS) \exists F \in \mathcal{F} : S' \text{ contains } F \iff A(S, PS, \mathcal{F}) = \text{unsafe}$$

One possibility to enforce termination is to add a “blur” operation as proposed in [SRW02]. This operation summarizes individuals which share the same unary-predicate values. In our setting we could apply this operation on each new structure we find, before adding it to the set of reachable shape graphs. By this the number of different shape graphs gets bounded and thus the number of shape graphs one can reach also gets bounded. Although this technique guarantees the termination of the algorithm, it also yields in a much coarser over-approximation of the set of reachable graphs.

5 Implementation

We implemented the verification algorithm as described in Sect. 4 in Java, making use of the source code of the well-known model checking tool TVLA
Fig. 10: Initial structure for turing machine simulation

Fig. 11: Helper shape production for turing machine simulation to simulate infinite long tape (left)

Fig. 12: Helper shape production for turing machine simulation to simulate infinite long tape (right)
Fig. 13: Production rule to simulate 'L' transitions of the turing machine

Fig. 14: Production rule to simulate 'R' transitions of the turing machine

Fig. 15: Forbidden graph for turing machine simulating. The predicate f represents an accepting state.
Thus we were able to take advantage of the already optimised code for shape graph transformations provided by TVLA.

Basically, our implementation loads a starting shape graph, a set of shape production rules, and a set of forbidden patterns, represented as text files each. Additionally, one needs to supply a text file listing the set core and instrumentation predicates, a set meaning functions for the instrumentation predicates, and a set of additional compatibility constraints (compatibility constraints derived from meaning functions are added automatically). The implementation then successively constructs the set of reachable shape graphs, each represented as logical structure, and checks whether a newly found shape graph contains one on the forbidden patterns. If the shape graph does contain a forbidden pattern, a counter example is generated that describes, how the shape graph was constructed as sequence of production applications. Otherwise, the shape graph is added to the set of reachable shape graphs and the maximum operation is applied. If no new shape graphs can be found anymore and none of the reachable shape graphs contains a forbidden pattern, the implementation asserts that the STS is safe.

We tested our implementation using the running example on a 3GHz Intel Core2Duo Windows System with 3GB main memory. Our implementation needs about 250ms to verify that the STS is safe. While doing so, it temporarily constructs 108 intermediate logical structures and finds 17 logical structures in the maximised set of reachable shape graphs. We also tested another example consisting of the starting shape graph shown in Fig. 16, the production rules EnterStation, LeaveStation, MoveSingle, PickUp (Fig. 17), and DropOff (Fig. 18), and the forbidden pattern shown in Fig. 19. There the task of the implementation is to find a sequence of productions such that person $P$ gets from station $s_1$ to station $s_2$. To find the minimal plan for this, shown in Fig. 20, the implementation needs about 1800ms, constructs 356 logical structures, and finds 102 logical structures in the maximised set of reachable shape graphs.

Both case studies show that the most expensive operation in terms of runtime is the $\text{max}$ operation. We implemented the $\text{max}$ operation by checking for each
Fig. 17: Production rule: PickUp

Fig. 18: Production rule: DropOff

Fig. 19: Forbidden pattern. Describes the goal situation of the second example STS.
Fig. 20: Counter example showing a path in the second example STS to a shape graph containing the forbidden pattern shown in Fig. 19.
newly found shape graph whether it can be embedded in a shape graph in the (current) set of reachable shape graph or vice versa. Thus, for each newly found shape graph we need $2n$ embedding checks, if $n$ denotes the number of shape graphs in the current set of reachable shape graphs. Furthermore, for arbitrary shape graphs checking for embedding is NP-complete ([AMSS06]). Hence it is not surprising that the max operation was observed to be very costly. Currently, we use a simple backtracking algorithm to check for embedding. TVLA also offers a more sophisticated algorithm described in [AMSS06] that relies on solving graph-matching problems. For our small examples, however, the backtracking algorithm turned out be more efficient. This may change, of course, for more complex examples.

6 Conclusion

In this paper, we have introduced a shape analysis approach for generating a finite over-approximation of the reach set of a graph transformation system with infinite state space. In contrast to some of the other work done in this area, e.g. [Ren04], we derive from our strict adherence to the formalism presented in [SRW02] a very straightforward avenue for implementation, which we have demonstrated using the 3-valued logic engine TVLA.

Combining this with forbidden graph patterns to model dangerous or undesirable situations, we have constructed a basic verification method for graph transformation systems with infinite state spaces. This verification method has been proven to be correct in the sense that it does not declare a graph transition system to be safe even though paths containing forbidden patterns exist (i.e. the algorithms does not produce false positives).

Because we have based our work on the parametric shape analysis introduced by Sagiv et.al. ([SRW02]), we can benefit from their notion of instrumentation, allowing us to tune the shape analysis to the relevant properties we want to check. Currently, this has to be done manually, but we hope to be able to provide an automatic construction of instrumentation predicates in the future.

Currently, there are a number of limitations to our approach. These limitations include the following:

**False Positives** Our abstraction does not guarantee that a counterexample that was found on the shape graph level can be concretized to a series of concrete graph transformations.

**Parallel Rule Application** Currently, all rules are assumed to be atomic and have to be applied consecutively, rather than concurrently

**Extended Graph Formalisms** We have not yet explored how our approach must be modified to accommodate the use of annotated or generalized graphs, such as typed graphs or multigraphs.
NACs There is currently no way to include negative application conditions in our graph transformation systems.

Transitive Closure Since performing potential global updates of formulae with transitive closure would violate the locality of rule application, transitive closure is currently not included in our approach.

In order to emphasize the qualities of our approach, we will now discuss how it relates to other work in this area.

In [BBKR08], a method for automatic abstraction of graphs is introduced. Intuitively, nodes are identified if their neighbourhood of radius $k \in \mathbb{N}$ is the same. While this automatic abstraction greatly reduces the need for human intervention in the verification process, it also reduces the flexibility of the approach. Only a certain class of systems can be handled well by neighbourhood abstraction, while our approach can be tuned to fit the needs of very different systems on a per-system basis. Furthermore, the method from [BBKR08] cannot use information from spurious counterexamples, since the abstraction leaves them with only one degree of freedom, the radius $k$. In contrast, using additional instrumentation predicates, our approach can utilise the full amount of information from spurious counterexamples.

Another approach to verifying infinite-state systems is the one by Saksena, Wibling and Jonsson [SWJ08]. It is based on backwards application of rules. By applying inverted rules to the forbidden patterns it is possible to determine whether a starting graph can lead to a failure state. The backwards application paradigm imposes some restrictions on this approach, for example forbidding the deletion of nodes and requiring a single starting pattern. Our approach does not suffer such restrictions. Furthermore, since the approach does not include an explicit abstraction and thus no information about the rest of the graph is available when applying a rule to a pattern, it would be very difficult to include concepts such as parameterised rules or parallel rule application. Since our approach uses explicit abstraction through shapes, it can encode information about the entire graph and is thus much more suited to support such extensions.

Lastly, Baldan, Corradini and König [BCK08] have written a series of papers in which they develop a unique approach to the verification of infinite state GTSs. They relate GTSs to Petri nets and construct a combined formalism, called a petri graph, on which they show certain properties via a technique called unfolding. This approach achieves many of the goals we strive for. However, a single concrete start graph is required for an analysis, which would be a major restriction in systems where there are many possible initial states, or even an unknown initial state.

The above discussion of related work is by no means exhaustive, but it suffices to show that, while each of these approaches has currently some advantages over our approach, no single approach outperforms ours in every single way. The results described in this paper lay the foundations for a new approach to the verification of infinite-state GTSs, which we strongly believe to be better
suited to overcome the many problems facing any theory in this area, than the currently available approaches. As such, there are a number of limitations to our approach which we intend to tackle in the future. We plan to look at parallel rule application, negative application conditions [HHT96] and especially rules with quantifiers [Ren06] which allow to specify changes on arbitrary numbers of nodes of some particular type within one rule.

7 Future Work

Motivated by the remaining shortcomings of our approach, we plan to expand and modify it to overcome them. A major field of interest here is the inclusion of parametrised rules in our verification framework. These would be specified by shape graphs and could represent an infinite number of similar, concrete rules. An example for this is a rule which specifies the movement of a RailCab convoy of arbitrary length.

As mentioned in Section 6, we plan to automatically generate sets of instrumentation predicates. We intend to do this by inspecting the structure of the graph pattern we wish to exclude from our graph transformation system. This would be compared to the structure of the graphs themselves, possibly derived from the core predicates. With the information thus gained, a set of properties which seem necessary for the identification of the forbidden pattern and are not mirrored by the core predicates could be constructed.

Since our approach can produce false positives, we will investigate two different approaches to overcome this problem.

In the first approach we will investigate whether an extended form of compatibility constraints, called deductive constraints, can be used to carry additional information through multiple production applications in order to eliminate false positives altogether.

The second, somewhat more traditional approach will introduce a CEGAR-like [CGJ00] approach to abstraction refinement to our shape analysis.

Lastly, we plan to study the effects of allowing parallel application of rules.

References


A Appendix

Proposition 4. Let $G = \langle N^G, E^G \rangle$, $G' = \langle N^{G'}, E^{G'} \rangle$ be graphs, $P = \langle L, R \rangle$ a graph production rule, $P' = (P, \gamma)$ the corresponding shape production rule, and
$m : N_L \rightarrow N_G$ an injective mapping. It holds:

$$G \xrightarrow{P, m} G' \Leftrightarrow \mathit{ls}(G) \xrightarrow{P', m} \mathit{ls}(G')$$

**Proof.**

“⇒” We first of all need to show that we can apply $P'$ to $\mathit{ls}(G)$. Thus, $m$ needs to be an injective matching and for all $(n, p, n') \in E_L$ it must hold: $\iota(p)(m(n), m(n')) = 1$ (w.l.o.g. $p$ binary). The former is true, because $m$ is injective by definition and because $U^{\mathit{ls}(G)} = N^G$ holds. Furthermore, we get:

$$(n, p, n') \in E_L \Rightarrow (m(n), p, m(n')) \in E^G \Leftrightarrow \iota(p)(m(n), m(n')) = 1$$

Hence, $P'$ is indeed applicable to $\mathit{ls}(G)$ and using the matching $m$ we get some shape $S \ (G, P', m, S)$. We are left to show, that $S = \mathit{ls}(G')$ holds. By construction, it holds $U^S = U^{\mathit{ls}(G')} = N^G$. Because, we are talking about two-valued lattice, we also get $\iota^S(sm) = 0 = \iota^{\mathit{ls}(G')}(sm)$. Finally, for some w.l.o.g. binary predicate $p$, we get:

$$\iota^S(p)(u_1, u_2) = \begin{cases} 
0 & \text{if } (u_1, p, u_2) \in m(E^-) \\
1 & \text{if } (u_1, p, u_2) \in \hat{m}(E^+) \\
\iota^{\mathit{ls}(G)}(p)(u_1, u_2) & \text{else}
\end{cases}$$

- If $(u_1, p, u_2) \in m(E^-)$, then the edge got removed in $G'$. Thus, we also have $\iota^{\mathit{ls}(G')} = 0$.
- If $(u_1, p, u_2) \in \hat{m}(E^+)$, the edge got added in $G'$. Thus, we also have $\iota^{\mathit{ls}(G')} = 1$.
- Otherwise:

$$\iota^{\mathit{ls}(G')} (u_1, u_2) = 1 \Leftrightarrow (u_1, p, u_2) \in E^{G'} \Leftrightarrow (u_1, p, u_2) \in E^G$$

$$\Leftrightarrow \iota^{\mathit{ls}(G)} (u_1, u_2) = 1 = \iota^S(p)(u_1, u_2)$$

“⇐” Again, we need to show, first of all, that $m$ can be applied to $G$. Thus, we need to show that $m$ is a morphism. It holds (w.l.o.g. $p$ binary):

$$(n, p, n') \in E_L \Rightarrow \iota(p)(m(n), m(n')) = 1 \Leftrightarrow (m(n), p, m(n')) \in E^G$$

Hence, $m(E_L) \subseteq E^G$ and $m$ is a morphism. We can thus apply $m$ to $G$ and get some graph $G''$. We are left to show that $G'' = G'$ holds. By construction, we have $N^{G''} = N^{G'} = U^{\mathit{ls}(G')}$. We distinguish some cases again $(u_1, u_2 \in U^{\mathit{ls}(G')}$ and w.l.o.g. binary):

- If $(u_1, p, u_2) \in m(E^-)$, then we get $\iota^{\mathit{ls}(G')} = 0$ and thus $(u_1, p, u_2) \notin E^{G'}$.
- By construction, we also have $(u_1, p, u_2) \notin E^{G''}$.
- If $(u_1, p, u_2) \in \hat{m}E^+$, we get $\iota^{\mathit{ls}(G')} = 1$ and thus $(u_1, p, u_2) \in E^{G'}$. By construction, we also have $(u_1, p, u_2) \in E^{G''}$.
Lemma 3. Let $G$ be a two-valued and $S'$ be a three-valued logical structure with $G \subseteq_f S'$. Let $\varphi$ be some formula. Let $m$ be an assignment that assigns the free variables of $\varphi$ to individuals of $G$. Let $f' : U^{S'} \to U^G$ be a the (injective) function with $f'(u) \in f^{-1}(u)$ for all $u \in U^{S'}$. For $v \in \{0, 1\}$, it holds:

\[
[\varphi]^{S'}_m = v \Rightarrow [\varphi]^{G}_{f'om} = v
\]

In other words, it holds:

\[
[\varphi]^{G}_{f'om} \subseteq [\varphi]^{S'}_m
\]

Proof. We show this by induction over the meaning semantics:

Atomic. For $l \in \{0, 1/2, 1\}$:

\[
[l]^{S'}_m = l = [l]^{G}_{f'om}
\]

For some $k$-ary predicate $p \in \mathcal{P}$ with free variables $v_1, \ldots, v_k$:

\[
[p(v_1, \ldots, v_k)]^{S'}_m = \iota^{S'}(p)(m(v_1), \ldots, m(v_k)) = \iota^G(p)(f'(m(v_1)), \ldots, f'(m(v_k)))
\]

Thus, $[p(v_1, \ldots, v_k)]^{S'}_m = v \Rightarrow [p(v_1, \ldots, v_k)]^{G}_{f'om} = v$ for $v \in \{0, 1\}$.

For an atomic formula $(v_1 = v_2)$ (with free variables $v_1, v_2$), we have $[v_1 = v_2]^{S'}_m = 1$ iff $m(v_1) = m(v_2)$ and $\iota^{S'}(sm)(m(v_1)) = 0$. In this case, clearly $[v_1 = v_2]^{G}_{f'om} = 1$ holds. Similar, we get $[v_1 = v_2]^{S'}_m = 0$ iff $m(v_1) \neq m(v_2)$, in which case also $f'(m(v_1)) \neq f'(m(v_2))$ holds, because $f'$ is injective.

Logical Connectives. For some formulae $\varphi_1$ and $\varphi_2$, we get

\[
[\varphi_1 \land \varphi_2]^{S'}_m = 1 \Leftrightarrow [\varphi_1]^{S'}_m = 1 \land [\varphi_2]^{S'}_m = 1
\]

By induction, we then get $[\varphi_1]^{G}_{f'om} = 1$ and $[\varphi_2]^{G}_{f'om} = 1$, and thus $[\varphi_1 \land \varphi_2]^{G}_{f'om} = 1$. Similar, we get:

\[
[\varphi_1 \lor \varphi_2]^{S'}_m = 0 \Leftrightarrow [\varphi_1]^{S'}_m = 0 \lor [\varphi_2]^{S'}_m = 0
\]

\[
[\neg \varphi_1]^{G}_{f'om} = 0 \lor [\varphi_2]^{G}_{f'om} = 0 \Leftrightarrow [\varphi_1 \land \varphi_2]^{G}_{f'om} = 0
\]

Analogical, we conclude:

\[
[\varphi_1 \lor \varphi_2]^{G}_{f'om} = 0 \Rightarrow [\varphi_1 \lor \varphi_2]^{G}_{f'om} = 0
\]

\[
[\neg \varphi_1]^{S'}_m = v \Rightarrow [\neg \varphi_1]^{G}_{f'om} = v
\]
Quantifiers For some formula \( \varphi \), we have \( \forall v_1 : \varphi \rceil^G_{m} = 1 \) iff for all \( u \in U^{S'} \) \( \| \varphi \|_{m|v_1 \rightarrow u}^{S'} = 1 \) holds. By IH it holds for all \( u \in U^{S'} \) \( \| \varphi \|_{f^{-1}om[v_1 \rightarrow u]}^{G} = 1 \) and thus we get \( \forall v_1 : \varphi \rceil^G_{f^{-1}om} = 1 \). Analogically, we get:

\[
\begin{align*}
\forall v_1 : \varphi \rceil^G_{m} &= 0 \Rightarrow \forall v_1 : \varphi \rceil^G_{f^{-1}om} = 0 \\
\exists v_1 : \varphi \rceil^G_{m} &= u \Rightarrow \exists v_1 : \varphi \rceil^G_{f^{-1}om} = u
\end{align*}
\]

\( \Box \)

**Lemma 4.** Let \( G \) be a two-valued and \( S \) a three-valued structure. Let \( \Sigma \) be a set of compatibility constraint. Let \( G \sqsubseteq S \) and \( G \models \Sigma \). Then \( \text{coerce}(S) \neq \bot \) holds.

**Proof.** Let \( f \) be the embedding function that embeds \( G \) in \( S \). \( \text{coerce}(S) \) can be nil only if there is a right side of some compatibility constraint \( \varphi \triangleright a \) that gets a definite value for some assignment \( m \) that is not equal to 1 during the execution of the algorithm, i.e., \( \| a \|_{m}^{S'} = 0 \) for some \( S' \) with \( G \sqsubseteq S' \). Furthermore, \( \varphi \) must evaluate to 1 on such \( S' \), i.e., \( \| \varphi \|_{m}^{S'} = 1 \). By Lemma 3 we get, that \( \varphi \) then must also evaluate to 1 on \( G \), i.e., \( \| \varphi \|_{f^{-1}om}^{G} \) for some injective function \( f' : U^{S'} \rightarrow U^{G} \) with \( f'(u) \in f^{-1}(u) \) for all \( u \in U^{S'} \). Similar, we get \( 0 = \| a \|_{m}^{S'} = \| a \|_{f^{-1}om}^{G} \). But, since \( G \) does not contain \( \varphi \triangleright a \), we get the contradiction by \( \| a \|_{f^{-1}om}^{G} \neq 0 \). \( \Box \)

**Definition 24.** Let \( \Sigma \) be a set of compatibility constraints. Let

\[
\Pi = (P_1, \gamma_1), (P_2, \gamma_2), \ldots, (P_k, \gamma_k)
\]

be a chain of shape productions rules (that remain a correct instrumentation). For logical structures \( S_0 \) and \( S_k \) we say \( S_0 \xrightarrow{\Pi} S' \) iff \( S_0 \xrightarrow{P_1} S_1 \), \( \text{coerce}(S_1) \xrightarrow{P_2} S_2 \), \ldots, \( \text{coerce}(S_{k-1}) \xrightarrow{P_k} S_k \).

**Lemma 5.** Let \( \Pi = (P_1, \gamma_1), (P_2, \gamma_2), \ldots, (P_k, \gamma_k) \) be a chain of production. Let \( S_0 \) be a three-valued and \( G_0 \sqsubseteq S_0 \) a two-valued logical structure. Let \( \Sigma \) be a set of compatibility constraints and \( G_0 \models \Sigma \), \( S_0 \models \Sigma \). If \( G_0 \xrightarrow{\Pi} G' \) with \( G' \models \Sigma \) then there exists a structure \( S' \) such that \( S_0 \xrightarrow{\Pi} S' \), \( G' \sqsubseteq S' \), and \( \text{coerce}(S') \neq \bot \).

**Proof.** We proof the Lemma by induction over the length \( k \) of the production chain:

**I.B.** For \( k = 1 \) we have \( G_0 \xrightarrow{(P_1, \gamma_1)} G' \) and \( G' \models \Sigma \). Each materialisation on \( G_0 \) would result in \( G_m \sqsubseteq G_0 \). Because \( G_0 \) is two-valued another structure can only be embedded in \( G_0 \) if the structure is isomorphic to \( G_0 \). Hence, it holds: \( G_0 \xrightarrow{(P_1, \gamma_1)} G' \). The claim follows by Theorem 1.

**I.S.** \( ((k-1) \rightarrow k) \) Let \( \Pi' := (P_1, \gamma_1), (P_2, \gamma_2), \ldots, (P_{k-1}, \gamma_{k-1}) \). Let \( G_{k-1} \) \( (G_{k-1} \models \Sigma) \) be a two-valued structures such that \( G_0 \xrightarrow{\Pi'} G_{k-1} \), \( \text{coerce}(G_{k-1}) \xrightarrow{(P_{k-1}, \gamma_{k-1})} \)}
$G'$. By I.H. there exists a structure $S_{k-1}$ such that $S_0 \xrightarrow{H'} S_{k-1}$, $G_{k-1} \subseteq S_{k-1}$, and $\text{coerce}(S_{k-1}) \neq \bot$. Thus, we get analogical to argumentation in I.B.: $G_{k-1} \xrightarrow{(P_k, \gamma_k)} G'$. By Theorem 1 there exists a $S'$ such that

$$\text{coerce}(S_{k-1}) \xrightarrow{(P_k, \gamma_k)} S', \quad G' \subseteq S', \quad \text{and} \quad \text{coerce}(S') \neq \bot.$$ 

By construction of $S_{k-1}$, $S_0 \xrightarrow{H} S'$ also holds. \hfill \square