Full Semantics Preservation in Model Transformation – A Comparison of Proof Techniques

Mathias Hülsbusch[1], Barbara König[1], Arend Rensink[2], Maria Semenyak[3], Christian Soltenborn[3], and Heike Wehrheim[3]

[1] Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, Germany
[2] Department of Computer Science, University of Twente, The Netherlands
[3] Institut für Informatik, Universität Paderborn, Germany

Abstract. Model transformation is a prime technique in modern, model-driven software design. One of the most challenging issues is to show that the semantics of the models is not affected by the transformation. So far, there is hardly any research into this issue, in particular in those cases where the source and target languages are different.
In this paper, we are using two different state-of-the-art proof techniques (explicit bisimulation construction versus borrowed contexts) to show bisimilarity preservation of a given model transformation between two simple (self-defined) languages, both of which are equipped with a graph transformation-based operational semantics. The contrast between these proof techniques is interesting because they are based on different model transformation strategies: triple graph grammars versus in situ transformation. We proceed to compare the proofs and discuss scalability to a more realistic setting.

1 Background

One of today's most promising approaches for building complex software systems is the Object Management Group's *Model Driven Architecture* (MDA). The core idea of MDA is to first model the target system in an abstract, platform-independent way, and then to refine that model step by step, finally producing platform-specific, executable code. The refinement steps are to be performed automatically using so-called *model transformations*; the knowledge needed for each refinement step is contained in the respective transformation.

As a consequence, in addition to the source model's correctness, the correctness of the model transformations is crucial for MDA; if they contain errors, the target system might be seriously flawed. But how to ensure the correctness of a model transformation? In this paper, we take a formal approach: We *prove* that the presented model transformation is *semantics preserving*, i.e., we prove that the behaviour of source and generated target model is equivalent (in a very strict sense, discussed below) for *every* source model we potentially start with.

As an example of a realistically sized case for which behavioural preservation is desirable, in [5] we have presented a model transformation from UML Activity Diagrams [20] (called AD below) to TAAL [12], a Java-like textual language. The choice of this case is motivated by two reasons:

– It involves a transformation from an abstract visual language into a more concrete textual one, and hence it perfectly fits into the MDA philosophy.
– The semantics of both the source and target language (AD and TAAL) have been formally specified by means of graph transformation systems ([8] and [12], resp.).

The latter means that every AD model and every TAAL program give rise to a *transition system* modelling its execution. This in turn allows the application of standard concepts from concurrency theory in order to compare the executions and to decide whether they are indeed equivalent or not. Our aim is eventually to show *weak bisimilarity* between the transition system of any Activity Diagram and that of the TAAL program resulting from its transformation. We call this *full* semantic preservation because weak bisimilarity is one of the most discriminating notions of behavioural equivalence.

Unfortunately, the size and complexity of the above problem are such that we have decided to first develop proof strategies for the intended result on a much more simplified version of the languages. In the current paper, we therefore apply the same question to two toy languages, inspired by AD and TAAL. Then, we solve the problem using two contrasting proof strategies. The contribution of this paper lies in developing these general strategies, carrying out the proofs for our example and afterwards comparing the strategies. Although simple, our example exhibits general characteristics of complex model-to-model transformations: different source- and target languages, different levels of granularity of operational steps in the semantics and different labellings of steps. Our two proof strategies present general approaches to proving semantics preservation of such model transformations.

The *first strategy* relies on a triple graph grammar-based definition of the model transformation (see [13,27]). Based on the resulting (static) triple graphs, we define an explicit bisimulation relation between the dynamic, run-time state graphs.

The *second strategy* relies instead on an in-situ definition of the model transformation and an extension of the operational semantics to the intermediate (hybrid) models. Using the theory of borrowed contexts (see [4]), we show that each individual model transformation step preserves the semantics.

The rest of the paper is structured as follows: Section 2 sets up a formal basis for the paper. Additionally, the source and target language and their respective semantics are introduced. Sect. 3 introduces the model transformation, in both versions mentioned above (triple graph grammar-based and in-situ). The actual proofs are worked out in Sections 4 and 5, respectively. Finally, Sect. 6 discusses and evaluates the results. Detailed proofs and additional information are contained in the extended version of this paper [10].

2 Definitions

2.1 Graphs and Morphisms

Definition 1 (Graph). *A graph is a tuple $G = \langle V, E, src, tgt, lab \rangle$, where V is a finite set of nodes, E a finite set of edges, $src, tgt \colon E \to V$ are source and target functions associating nodes with every edge, and $lab \colon E \to \mathsf{Lab}$ is an edge labelling function. We always assume $V \cap E = \emptyset$.*

For a given graph G, we use V_G, E_G etc. to denote its components. Note that there is a straightforward (component-wise) definition of union and intersection over graphs, with the caveat that these operators may be undefined if the source, target or labelling functions are inconsistent.

In example graphs, we use the convention that self-edges may be displayed through node labels. That is, every node label in a figure actually represents an edge from that node to itself, with the given label. We now define morphisms as structure-preserving maps between graphs.

Definition 2 (Morphism). *Given two graphs G, H, a morphism $f\colon G \to H$ is a pair of functions $(f_V\colon V_G \to V_H, f_E\colon E_G \to E_H)$ from the nodes and edges of G to those of H, respectively, which are consistent with respect to the source and target functions of G and H in the sense that*

$$src_H \circ f_E = f_V \circ src_G \qquad tgt_H \circ f_E = f_V \circ tgt_G \qquad lab_H \circ f_E = lab_G \ .$$

If both f_V and f_E are injective (bijective), we call f injective (bijective).

A bijective morphism is often called an *isomorphism*: if there exists an isomorphism from G to H, we call them *isomorphic*.

A frequently used notion of graph structuring is obtained by *typing* graphs over a fixed *type graph*.

Definition 3 (Typing). *Given two graphs G, T, the graph G is said to be* typable over *T if there exists a* typing morphism *$t\colon G \to T$. A* typed graph *is a graph G together with such a typing morphism, say t_G. Given two graphs G, H typed over the same type graph (using typing morphisms t_G and t_H), a* typed graph morphism *$f\colon G \to H$ is a morphism that preserves the typing, i.e., such that $t_G = t_H \circ f$.*

Besides imposing some structural constraints over graphs, typing also provides an easy way to restrict to subgraphs:

Definition 4 (Type restriction). *Let T, U be graphs such that $U \subseteq T$, and let G be an arbitrary graph typed over T via $t : G \to T$. The* restriction of *G* to *U, denoted $\pi_U(G)$, is defined as the graph H such that*

- $V_H = \{v \in V_G \mid t(v) \in V_U\}$, $E_H = \{e \in E_G \mid t(e) \in E_U\}$,
- $src_H = src_G \restriction_{E_H}$, $tgt_H = tgt_G \restriction_{E_H}$ *and* $lab_H = lab_G \restriction_{E_H}$.

The set of graphs with the associated morphisms form a category, which we will denote by Graph.

2.2 Graph Languages

In this paper we consider model transformation between two languages. In particular, we consider *graph languages*, i.e. sets of graphs; the models are the graphs themselves. We concentrate on a running example where there are two distinct, very simple graph languages denoted $\mathcal{A}$ and $\mathcal{B}$. Fig. 1 shows type graphs for the languages, denoted $T^{\mathsf{st}}_{\mathcal{A}}$ and $T^{\mathsf{st}}_{\mathcal{B}}$, respectively. They describe the typing of the *static* parts of our two languages.

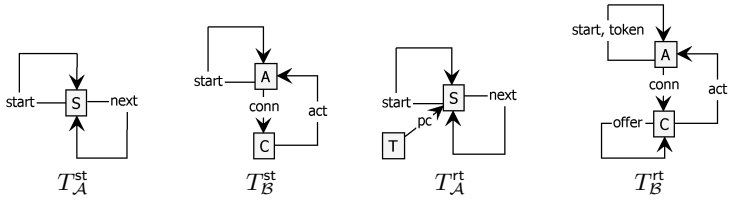


Fig. 1. Static (st) and run-time(rt) type graphs for graph languages $\mathcal{A}$ and $\mathcal{B}$.

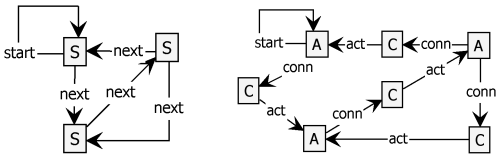


Fig. 2. Example graphs of languages $\mathcal{A}$ (left) and $\mathcal{B}$ (right).

We will sometimes also call these the (static) metamodels of the two languages. The figure also shows the corresponding extended *run-time* type graphs, which will be discussed below (Section 2.4).

The type graphs themselves impose only weak structure: not all graphs that can be typed over the $\mathcal{A}$- and $\mathcal{B}$-type graphs are considered to be part of the languages. Instead, we impose the following further constraints on the static structure:

Language $\mathcal{A}$ consists of next-connected S-labelled nodes (*statements*). There should be a single S-node with a start-edge to itself, from which all other nodes are reachable (via paths of next-edges). Furthermore no next-loops are allowed.

Language $\mathcal{B}$ consists of bipartite graphs of A- (*action*) and C-labelled (*connector*) nodes. Every C-node has exactly one incoming conn-edge and exactly one outgoing act-edge; the opposite nodes of those edges must be distinct. Like $\mathcal{A}$-graphs, $\mathcal{B}$-graphs have exactly one node with a start-self-edge, from which all other nodes are reachable (via paths of conn- and act-edges).

Small example graphs are shown in Fig. 2. We use $\mathcal{G}^{st}_{\mathcal{A}}$ ($\mathcal{G}^{st}_{\mathcal{B}}$) to denote the set of all well-formed (static) $\mathcal{A}$-graphs ($\mathcal{B}$-graphs).

2.3 Rules and Rule Systems

To specify the semantics of our languages, we have to formally describe changes on our graphs. This is done by means of *graph transformation rules*. A rule describes the change of (parts of) a graph by means of a before and after template (the left-hand and right-hand hand side of a rule); the interface fixes the part on which left and right hand side have to agree.

Definition 5 (Transformation rule). *A graph transformation rule is a tuple* $r = \langle L, I, R, \mathcal{N} \rangle$*, consisting of a left hand side (LHS) graph* L*, an interface graph* I*, a right hand side (RHS) graph* R*, and a set* $\mathcal{N} \subseteq$ Graph *of* negative application

conditions (NAC's)*, which are such that* $L \subseteq N$ *for all* $N \in \mathcal{N}$*. The interface* I *is the intersection of* L *and* R *(*$I = L \cap R$*).*

We let Rule denote the set of rules. A rule (without a NAC) is basically a pair of injective morphisms in Graph: $L \leftarrow I \rightarrow R$. The diagram for a rule with NACs is this basic span together with the injective morphisms from L to the elements of $\mathcal{N}$. For a single NAC N, a rule has the following form: $N \leftarrow L \leftarrow I \rightarrow R$. There are other definitions of graph-transformation rules in the literature, the one used here is the one for double-pushout rewriting (DPO-rewriting).

A transformation rule $r = \langle L, I, R, \mathcal{N} \rangle$ is *applicable* to a graph G (called the *host graph*) if there exists an injective *match* $m\colon L \rightarrowtail G$ such that for no $N \in \mathcal{N}$ there exists a match $n\colon N \rightarrow G$ with $m = n \restriction_L$ (i.e., all negative application conditions are *satisfied*), and moreover, the *dangling edge condition* holds: for all $e \in E_G$, $src(e) \in m(V_L \setminus V_I)$ or $tgt(e) \in m(V_L \setminus V_I)$ implies $e \in m(E_L \setminus V_I)$. This condition can be understood by realising that the elements of G that are in $m(L)$, but not in $m(I)$, are scheduled to be deleted by the rule, whereas the elements in $m(I)$ are preserved (see below). Hence we can not delete a node without explicitly deleting all adjacent edges.

Given such a match m, the *application* of r to G is defined by extending m to $L \cup R$, by choosing distinct "fresh" nodes and edges (outside V_G and E_G, respectively) as images for $V_R \setminus V_L$ and $E_R \setminus E_L$ and adding those to G. This extension results in a morphism $\bar{m}\colon (L \cup R) \rightarrow C$ for some extended graph $C \supseteq G$. Now let H be given by

$$V_H = V_C \setminus m(V_L \setminus V_R), \qquad E_H = E_C \setminus m(E_L \setminus E_R)$$

together with the obvious restriction of src_C, tgt_C and lab_C to E_H. H is called the *target* of the rule application; we write $G \xrightarrow{r,m} H$ to denote that m is a valid match on host graph G, giving rise to target graph H, and $G \xrightarrow{r} H$ to denote that there is a match m such that $G \xrightarrow{r,m} H$. Note that H is not uniquely defined, due to the freedom in choosing the fresh images for $V_R \setminus V_L$ and $E_R \setminus E_L$; however, it is well-defined up to isomorphism.

Definition 6 (Rule system). *A rule system is a partial mapping* $\mathcal{R}\colon \mathsf{Sym} \rightharpoonup \mathsf{Rule}$*. Here,* Sym *is a universe of* rule names*.*

2.4 Language Semantics

In the context of the two languages defined in Section 2.2, we can use graph transformation rules for two separate purposes: to give a grammar that precisely and formally defines the languages or to specify the operational language semantics. In the latter case, the transformation rules describe patterns of state changes.

We will demonstrate the second usage here, by giving operational rules for $\mathcal{A}$-graphs and $\mathcal{B}$-graphs. This means that the graphs will represent run-time states. As we will see, this will involve auxiliary node and edge types that do not occur in the language type graphs. Fig. 1 shows extended type graphs $T^{\mathsf{rt}}_{\mathcal{A}}$ and $T^{\mathsf{rt}}_{\mathcal{B}}$ that include these *run-time* types. For $\mathcal{A}$, a T-node (of which there can be at most one) models a *thread*, through a single program counter (pc-labelled edge). For $\mathcal{B}$, we use token- and offer-loops which play a similar role; details will become clear below. Similar to the static

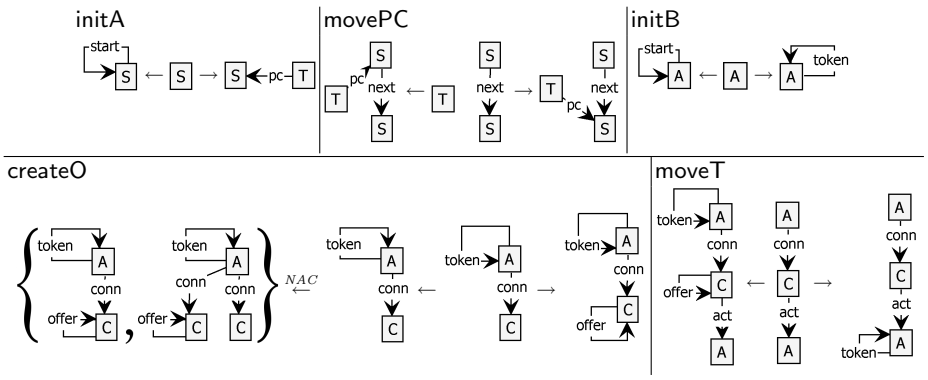


Fig. 3. Operational rules for $\mathcal{A}$ (initA and movePC) and $\mathcal{B}$ (initB, createO and moveT).

part, we use $\mathcal{G}_A^{\mathsf{rt}}$ ($\mathcal{G}_B^{\mathsf{rt}}$) to denote the set of well-formed (run-time) $\mathcal{A}$-graphs ($\mathcal{B}$-graphs). The semantics of $\mathcal{A}$- and $\mathcal{B}$-models are defined in Fig. 3. Note that the figure shows the rules in DPO style, i.e. the middle part gives the interface I, and the sides are L and R, given as $L \leftarrow I \rightarrow R$.

We let $dom(\mathcal{R}_\mathcal{A}) = \{\mathsf{initA}, \mathsf{movePC}\}$ and $dom(\mathcal{R}_\mathcal{B}) = \{\mathsf{initB}, \mathsf{createO}, \mathsf{moveT}\}$ be the names in the rule systems for the $\mathcal{A}$- and $\mathcal{B}$-models, the mapping to rules follows Fig. 3. Intuitively, the init-rules perform an initialisation of the run-time system, setting the program counter to the start statement (in $\mathcal{A}$) or putting a token onto a start action (in $\mathcal{B}$). Rule movePC simply moves the program counter to the next statement, createO moves an offer to a C-node and moveT moves the token. The semantics of $\mathcal{A}$- and $\mathcal{B}$-graphs is completely fixed by these rules, giving rise to a labelled transition system summarizing all these executions.

Definition 7 (Labelled transition system). *An* L-labelled transition system *(LTS) is a structure* $S = \langle Q, \rightarrow, \iota \rangle$*, where* Q *is a set of states and* $\rightarrow \subseteq Q \times L \times Q$ *is a set of transitions labelled over some set of labels* L*. Furthermore* $\iota \in Q$ *is the start state.*

In our case, the states are graphs and the transitions are rule applications. That is, given a rule system $\mathcal{R}$ and a start graph G, we obtain a $dom(\mathcal{R})$-labelled transition system by recursively applying all rules to all graphs. We will denote this transition system by $S(G)$ (leaving the rule system $\mathcal{R}$ implicit). For instance, the LTS of an $\mathcal{A}$-graph G is $S(G) = (\mathcal{G}_\mathcal{A}^{\mathsf{rt}}, \rightarrow_\mathcal{A}, G)$, where $\rightarrow_\mathcal{A}$ is defined by the rules in $\mathcal{R}_A$.

Semantic equivalence comes down to equivalence of the LTSs generated by two different graphs. There are several notions of equivalence over LTSs; see, e.g., [28]. In this paper, we use *weak bisimulation*. Weak bisimulation requires two states to mutually simulate each other, where a simulation may however involve internal (unobservable) steps. As usual, we use the special transition label τ to denote such internal steps.

For states $q, q' \in Q$ and a label α, we write $q \stackrel{\alpha}{\Longrightarrow} q'$ if $q \stackrel{\tau}{\rightarrow}^* \stackrel{\alpha}{\rightarrow} \stackrel{\tau}{\rightarrow}^* q'$ and use $\stackrel{\varepsilon}{\Longrightarrow}$ to stand for $\stackrel{\tau}{\rightarrow}^*$. Furthermore, we define for (visible or invisible) labels α the following function $\widehat{\ }$: $\widehat{\tau} = \epsilon$ and $\widehat{\alpha} = \alpha$ if $\alpha \neq \tau$.

Definition 8 (Weak bisimilarity). *Weak bisimilarity between two labelled transition systems S_1, S_2 is a relation $\approx \subseteq Q_1 \times Q_2$ such that whenever $q_1 \approx q_2$*

- *If $q_1 \xrightarrow{\alpha} q_1'$, then $q_2 \stackrel{\widehat{\alpha}}{\Longrightarrow} q_2'$ such that $q_1' \approx q_2'$;*
- *If $q_2 \xrightarrow{\alpha} q_2'$, then $q_1 \stackrel{\widehat{\alpha}}{\Longrightarrow} q_1'$ such that $q_1' \approx q_2'$.*

We call S_1 and S_2 as a whole weakly bisimilar, denoted $S_1 \approx S_2$, if there exists a weak bisimilarity relation between S_1 and S_2 such that $\iota_1 \approx \iota_2$.

2.5 Semantics-preserving Model Transformation

Our objective is to compare the LTSs of graphs of languages $\mathcal{A}$ and $\mathcal{B}$. In Section 3 we will define a (relational) model transformation $MT \subseteq \mathcal{G}^{\mathsf{st}}_{\mathcal{A}} \times \mathcal{G}^{\mathsf{st}}_{\mathcal{B}}$ translating $\mathcal{A}$-graphs to $\mathcal{B}$-graphs. We aim at proving this model transformation to be *semantics preserving*, in the sense that the LTSs of source and target models are always weakly bisimilar.

However, there is an obvious problem: the LTSs of $\mathcal{A}$- and $\mathcal{B}$-graphs do not have the same labels, in fact $dom(\mathcal{R}_{\mathcal{A}}) \cap dom(\mathcal{R}_{\mathcal{B}}) = \emptyset$. Nevertheless, there is a clear intuition which rules correspond to each other: on the one hand the two initialisation rules, and on the other hand the rules movePC and createO. The reason for taking the latter two as corresponding is that both rules decide on where control is moving. The rule moveT has no matching counterpart in the $\mathcal{A}$-language, it can be seen as an *internal* step of the $\mathcal{B}$-language, completing a step initiated by createO.

These observations give rise to the following approach: we rename the labels of the LTSs to be compared (i.e., the rule names) to a common set of names.

$$\begin{array}{lllll} map_{\mathcal{A}}: & \mathsf{initA} \mapsto \mathsf{init}, & \mathsf{movePC} \mapsto \mathsf{move} & & \\ map_{\mathcal{B}}: & \mathsf{initB} \mapsto \mathsf{init}, & \mathsf{createO} \mapsto \mathsf{move}, & \mathsf{moveT} \mapsto \tau & \end{array}$$

Definition 9 (Preservation of semantics). *Given two (graph) languages $\mathcal{G}^{st}_{\mathcal{A}}, \mathcal{G}^{st}_{\mathcal{B}}$, a model transformation $MT \subseteq \mathcal{G}^{st}_{\mathcal{A}} \times \mathcal{G}^{st}_{\mathcal{B}}$ is* semantics-preserving *if there are mapping functions $map_{\mathcal{A}}, map_{\mathcal{B}}\colon \mathsf{Sym} \to \mathsf{Sym}$ such that for all $G_A \in \mathcal{G}^{st}_{\mathcal{A}}$, $G_B \in \mathcal{G}^{st}_{\mathcal{B}}$ with $MT(G_A, G_B)$*

$$map_{\mathcal{A}}(S(G_A)) \approx map_{\mathcal{B}}(S(G_B)) \ .$$

3 Model Transformation

Our model transformation needs to translate $\mathcal{A}$-models into $\mathcal{B}$-models. We will actually present two definitions of the transformation, both tailored towards the specific proof technique used for showing semantics preservation.

3.1 Triple graph grammars

Our first transformation uses triple graph grammars (TGGs) which are well-suited for defining model transformations. TGG rules [27,13] typically describe the transformation between models of *different* types. The main idea is that the graphs used therein

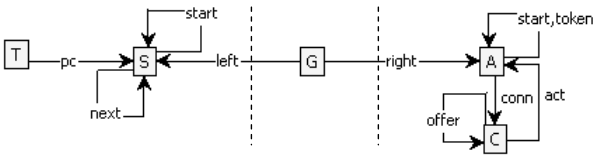


Fig. 4. Type graph $T^{\mathsf{rt}}_{\mathcal{AB}}$ for TGG graph rules

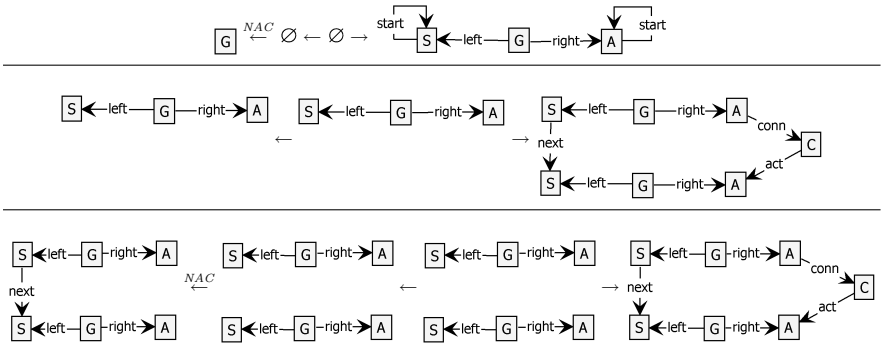


Fig. 5. TGG transformation rules

can be separated into three subgraphs, each being typed over its own type graph. Two of these subgraphs evolve simultaneously while the third keeps correspondences between them. For our example, we have the two type graphs $T^{\mathsf{rt}}_{\mathcal{A}}$ and $T^{\mathsf{rt}}_{\mathcal{B}}$ which - for forming a type graph for TGGs - are conjoined and augmented with one new correspondence G-node (the glue) (see Fig. 4). This combined type graph is denoted $T^{\mathsf{rt}}_{\mathcal{AB}}$.

Normally, for a transformation, the source model is given in the beginning and is then gradually transformed. TGG rules however build two models simultaneously, matching each part of the source model to the target one. This allows to keep correspondences between transformed elements and to prove certain properties of the corresponding graphs. The TGG rules for the $\mathcal{A}$ to $\mathcal{B}$ transformation are given in Fig. 5.

These rules incrementally build combined $\mathcal{A}$ and $\mathcal{B}$-graphs. Initially, only the upper rule in Fig. 5 can be applied and its application constructs a graph with one S- and one A-node connected via one correspondence node. The middle rule allows to create further S, A and C-nodes together with their correspondences, and the lower rule simultaneously generates new next-edges between S-nodes and connections between A-nodes via C nodes, however only for *corresponding* S- and A-nodes. We let $\mathcal{G}^{\mathsf{rt}}_{AB}$ denote the set of graphs obtained by applying the three TGG rules on an empty start graph. To obtain the translation at the end, we need to project the final graph onto the type graphs of $\mathcal{A}$ and $\mathcal{B}$. Using the definition of type restriction as given in Section 2, the model transformation MT thus works as follows: Given an $\mathcal{A}$-graph G_A and a $\mathcal{B}$-graph G_B, we have $MT(G_A, G_B)$ exactly if there is some $G_{AB} \in \mathcal{G}^{\mathsf{rt}}_{AB}$ such that $G_A = \pi_{T^{\mathsf{st}}_{\mathcal{A}}}(G_{AB})$ and $G_B = \pi_{T^{\mathsf{st}}_{\mathcal{B}}}(G_{AB})$.

3.2 In-situ Transformation

Instead of building two models simultaneously, in-situ transformations destroy the source model while building the target model. They have the disadvantage of leading to "mixed" states which incorporate components of both the source and the target model. This necessitates additional operational rules (see Section 5). On the other hand, an in-situ transformation describes a clear evolution process. Hence we currently find it better suited as a basis for our proof strategy 2 which relies on a congruence result for bisimilarity, which we use to show that replacing a part of the model does not modify behavioural equivalence for the entire model.

We will now present the in-situ transformation rules, which are shown in Fig. 6. The first rule relabels nodes by replacing the label S by the label A[4]. The second rule replaces a next-edge by a connection via a C-node. The third rule replaces the program counter by a token and allows the transformation of run-time models. We have reached a model in language $\mathcal{B}$ as soon as no further rule applications are possible. We define that $MT(G_A, G_B)$ iff G_A is transformed into G_B via the rules in Fig. 6.

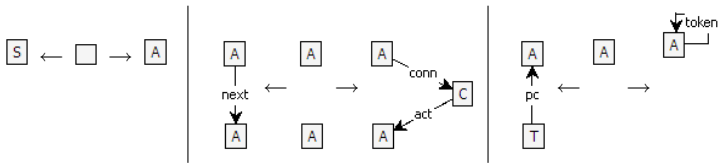


Fig. 6. In-situ transformation rules from language $\mathcal{A}$ to language $\mathcal{B}$.

3.3 Comparison

In this section we argue that both strategies define the same model transformation. Assume that a graph G_A is transformed into a graph G_B via the TGG transformation of Section 3.1. This means that G_A and G_B are constructed simultaneously by the TGG grammar and arise as projections of a graph G_{AB}. Then we can apply the in-situ rules of Fig. 6 to G_A, obtaining the corresponding items of G_B.

The other direction is slightly more complicated. Assume that we are given a graph G_A of language $\mathcal{A}$. Then, with the TGG rules, we generate a graph G_{AB} which projects (via $\pi_{T^{\text{st}}_{\mathcal{A}}}$) to G_A. We can then show, by induction on the length of this generating sequence and by using the fact that the transformation rules are confluent, that the graph $\pi_{T^{\text{st}}_{\mathcal{B}}}(G_{AB})$ obtained in this way coincides with G_B, the graph generated by applying the in-situ transformation rules as long as possible.

4 Proof Strategy 1

In this section, we present our first approach to proving semantic preservation of the model transformation on all source models (more details are given in Appendix A).

[4] Remember that labels are represented by loops on an unlabelled node.

This proof strategy uses the correspondences generated by the TGG rules, despite the fact that the semantic rules are applied on the individual models, based on the following two observations.

First observation Both for $\mathcal{A}$ and $\mathcal{B}$-models, the operational rules keep the syntactic, static structure of a model, except for start-edges: all S-nodes and next-edges, and all A, C-nodes and conn, act-edges stay the same.

To formulate structural correspondences, we introduce the following notation. For an S-node v_S and an A-node v_A, we write $corr(v_S, v_A)$ if there is a G-node v_G and a left-edge from v_S to v_G and a right-edge from v_G to v_A. For an edge e labelled label going from a node v to v', we simply write $\mathsf{label}(v, v')$. We also use these as predicates. The first result shows that correspondences between S and A-nodes are unique. Here, $\exists!$ stands for "there exists exactly one".

Proposition 10. *Let $G \in \mathcal{G}^{rt}_{AB}$, v_S an S-node and v_A an A-node in G. Then the following two properties hold: (A) $\exists! v$ of type A such that $corr(v_S, v)$, and (B) $\exists! v$ of type S such that $corr(v, v_A)$.*

A number of further results show that (1) corresponding nodes either both or none have start-edges, and (2) next-edges between S-nodes will generate connections via C-nodes between corresponding A-nodes and vice versa.

Second observation Correspondences between nodes in $\mathcal{A}$-models and $\mathcal{B}$-models are kept during application of semantic rules. Predicate $corr$ as well as Prop. 10 and properties (1) and (2) can thus also be applied to separate $\mathcal{A}$ and $\mathcal{B}$-graphs.

These two observations are crucial parts of our proof of semantic preservation.

Theorem 11. *Let G^0_A, G^0_B be an $\mathcal{A}$- and a $\mathcal{B}$-graph such that $MT(G^0_A, G^0_B)$. Then*

$$map_A(S(G^0_A)) \approx map_B(S(G^0_B))$$

For the proof, we need to construct a weak bisimulation relation $\mathcal{R}$ (defining $\approx$) between the states of the first and the second LTS:

$\mathcal{R} = \{(G_A, G_B) \in \mathcal{G}^{rt}_{\mathcal{A}} \times \mathcal{G}^{rt}_{\mathcal{B}} \mid \exists G_{AB} \in \mathcal{G}^{rt}_{\mathcal{AB}}$
(1) $\pi_{\mathsf{T}^{\mathsf{st}}_{\mathsf{A}} \setminus \mathsf{start}}(G_A) = \pi_{\mathsf{T}^{\mathsf{st}}_{\mathsf{A}} \setminus \mathsf{start}}(G_{AB}) \wedge \pi_{\mathsf{T}^{\mathsf{st}}_{\mathsf{B}} \setminus \mathsf{start}}(G_B) = \pi_{\mathsf{T}^{\mathsf{st}}_{\mathsf{B}} \setminus \mathsf{start}}(G_{AB})$,
(2) $\forall$ S-nodes v_S in G_A, A-nodes v_A in G_B s.t. $corr(v_S, v_A)$:
$\mathsf{start}(v_S)$ iff $\mathsf{start}(v_A)$,
(3) $\forall$ S-nodes v_S in G_A, A-nodes v_A in G_B s.t. $corr(v_S, v_A)$: $\exists v_T$ with $\mathsf{pc}(v_T, v_S)$ iff
(i) $\mathsf{token}(v_A) \wedge \forall v_C$ s.t. $\mathsf{conn}(v_A, v_C) : \neg\mathsf{offer}(v_C)$ or
(ii) $\neg\mathsf{token}(v_A) \wedge \exists v_C, v'_A : \mathsf{token}(v'_A) \wedge \mathsf{offer}(v_C) \wedge$
$\mathsf{conn}(v'_A, v_c) \wedge \mathsf{act}(v_C, v_A)$,

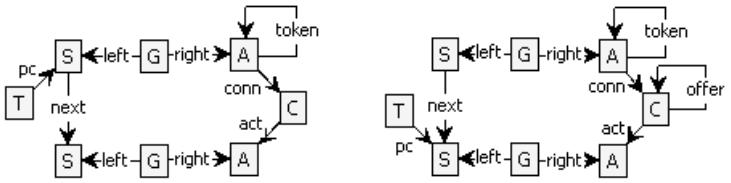


Fig. 7. Illustration of condition (3): Left (i), right (ii).

$$
\begin{aligned}
(4)\ \exists v_T, v_S : \mathsf{pc}(v_T, v_S) \iff\ & \neg\exists v'_S : \mathsf{start}(v'_S) \wedge \\
& \exists v_A : \mathsf{token}(v_A) \iff \neg\exists v'_A : \mathsf{start}(v'_A) \wedge \\
& \neg\exists v_A : \mathsf{start}(v_A) \implies \exists! v'_A : \mathsf{token}(v'_A) \wedge \\
& \forall v_C : \mathsf{offer}(v_C) \implies \exists v_A : \mathsf{token}(v_A) \wedge \mathsf{conn}(v_A, v_C) \wedge \\
& \neg\exists v_S : \mathsf{start}(v_S) \implies \exists! v'_S \text{ s.t. } \exists v_T : \mathsf{pc}(v_T, v'_S)\ \}
\end{aligned}
$$

It contains all pairs of $\mathcal{A}$ and $\mathcal{B}$-graphs which (1) in their static structure (except for start) still follow the structure generated by the TGG rules, (2) have start-edges only on corresponding nodes, (3) exhibit run-time properties only on corresponding nodes, and (4) obey certain well-formedness criteria for run-time elements.

Fig. 7 further illustrates condition (3). We have two possibilites for run-time elements in matching states: either the pc-edge is on an S-node and the token is on the corresponding A-node and no further offers exist (left), or the pc-edge is on a node for which the corresponding A-node has no token yet, but an offer has already been created and is ready to move the token to the A-node by means of the invisible step moveT (right). We show that the relation $\mathcal{R}$ is a weak bisimulation by proving that the states of transition systems can mimic each others moves. Due to space limitations we cannot give the full proof here, which can instead be found in the extended version [10].

5 Proof Strategy 2

5.1 The Borrowed Context Technique

In the following we will describe a different proof strategy, based on the borrowed context technique [4,23], which refines a labelled transition system (or even unlabelled reaction rules) in such a way that the resulting bisimilarity is a congruence (see also [15]). Weak bisimilarity as in Def. 8 is usually not a congruence. By a congruence we mean a relation over graphs that is preserved by contextualization, i.e., by gluing with a given environment graph over a specified interface. This is a mild generalization of standard graph rewriting in that we consider "open" graphs, equipped with a suitable interface.

The basic idea behind the borrowed context technique is to describe the possible interactions with the environment. In addition to existing labels, we add the following information to a transition: what is the (minimal) context that a graph with interface needs to evolve? More concretely we have transitions of the form

$$(J \to G) \overset{\alpha,(J \to F \leftarrow K),\mathcal{N}}{\longrightarrow} (K \to H)$$

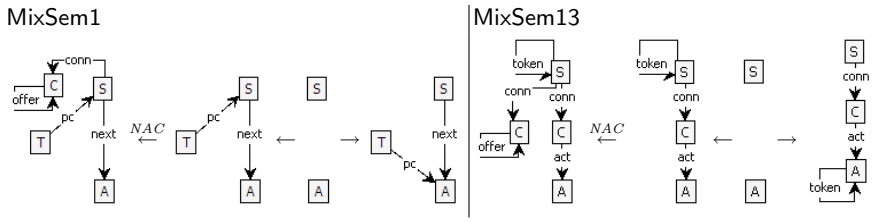


Fig. 8. Some rules of the operational semantics of mixed models

where the components have the following meaning: $(J \to G)$ is the original graph with interface J (given by an injective morphism from J to G) which evolves into a graph H with interface K. The label is now composed of three entities: the original label $\alpha = map(r)$ stemming from the operational rule r (as detailed in Section 2.5) and furthermore two injective morphisms $(J \to F \leftarrow K)$ detailing what is borrowed from the environment. The graph F represents the additional graph structure, whereas J, K are its inner and the outer interface. Finally we provide a set $\mathcal{N}$ of negative borrowed contexts, describing negative constraints on the environment (see also [23]). We are using a saturated and weak version of bisimulation (see Appendix B).

5.2 Using the Borrowed Context Technique for the Verification of Model Transformations

For in-situ model transformation within the same language, applications of the borrowed context technique are straightforward: show for every transformation rule that the left-hand and right-hand sides L, R with interface I are bisimilar with respect to the operational rules. Then the source model must be bisimilar to the target model by the congruence result. This idea has been exploited in [24] for showing behaviour preservation of refactorings.

However, in order to apply the idea above in our situation it is necessary to have an operational semantics also for "mixed" (or hybrid) models which incorporate components of both the source and the target model. Hence below we introduce such a mixed operational semantics, which has to satisfy the following conditions: (i) the mixed rules are *not* applicable to a pure source or target model; (ii) it is possible to show bisimilarity of left-hand and right-hand sides of all transformation rules. Finally, observe that our final aim is to show bisimilarity of closed graphs, i.e., of graphs with empty interface. It can be shown that if all left-hand sides are connected, the notion of bisimilarity induced by borrowed contexts coincides with the standard one.

5.3 Rules of the Mixed Semantics

There are sixteen additional rules for the mixed semantics. Seven of them handle the behaviour of pc-edges at A-nodes, seven the semantic of the token-edge at an S-node and two of them are mixed counterparts to the initialization rules. Fig. 8 shows two

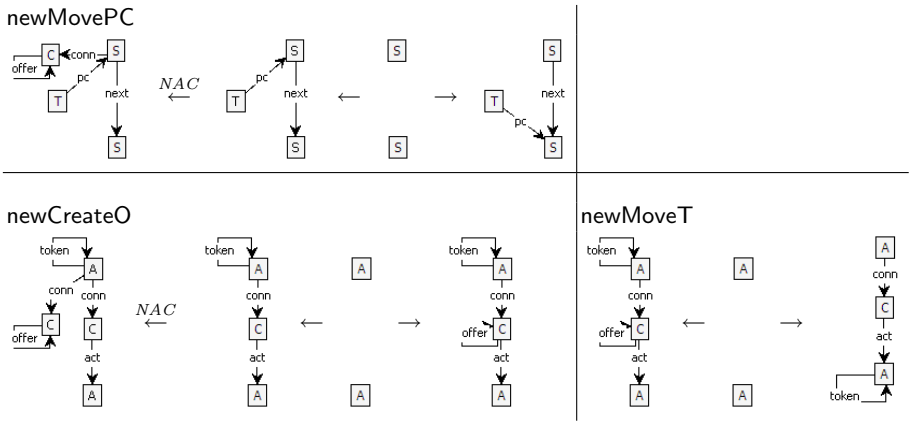


Fig. 9. Modified operational rules for the source and target languages

examples of mixed rules, the rest are provided in Appendix B. Here we work with a single function map (see Section 2.5), both rules in Fig. 8 are mapped to move.

Furthermore, we modify some of the operational rules of Fig. 3: first, we equip several rules, also of the source semantics (language $\mathcal{A}$) with NACs (without changing the operational behaviour). Second, we restrict to a minimal interface by deleting and recreating the connections (see Fig. 9). Due to the layout of the graphs, this does not modify the semantics. Both modifications are needed to make the proof work and the latter modification is also very convenient since it allows us to derive fewer labels.

5.4 The In-situ Transformation Preserves Weak Bisimilarity

Theorem 12. *The left-hand sides and right-hand sides of the three in-situ transformation rules in Fig. 6 are weakly bisimular, with respect to the borrowed contexts technique, under the rules of the mixed semantics.*

Since weak bisimilarity is a congruence (see Appendix B) and borrowed context bisimilarity coincides with standard bisimilarity (see Def. 8) on source and target models, this implies that $map(S(G_A)) \approx map(S(G_B))$ *whenever* $MT(G_A, G_B)$.

We give some intuition on the label derivation process by discussing one example, which needs the handling of weak moves and NACs (see Fig. 10).

In the labelled transition system, the graph consisting only of an S-node makes a move (with rule MixSem13) with the label shown in the (big) dashed box, i.e., it borrows a token, a C-node and an A-node. Spelling out the transition labels more concretely we have $\alpha =$ move, F is the graph in the dashed box on the left (where the grey node represents both interfaces J, K) and the only NAC in $\mathcal{N}$ is given on the right. The corresponding graph (the A-node) can answer this step with the same label, by making a step with rule newCreateO plus a weak step (τ) with rule newMoveT. After this second step, using an up-to-context proof technique, the same context (see dotted boxes) can be removed from both graphs, leaving the original pair of graphs already in the relation.

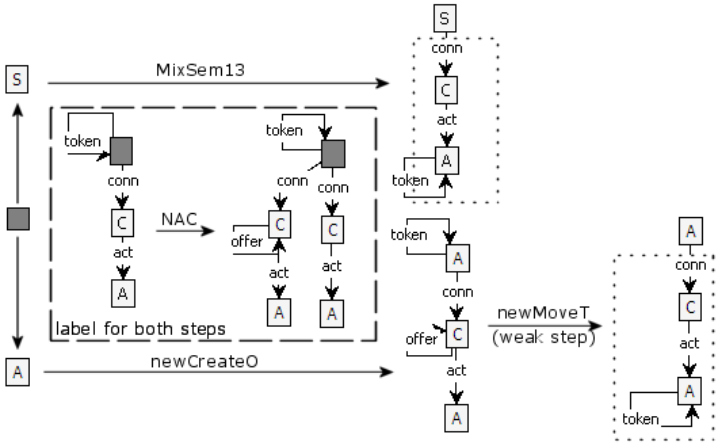


Fig. 10. Example of a label derivation using the borrowed context technique

On the other hand, the answer to the newCreateO-step is with rule MixSem13. So the pair of graphs reached after one step has to be in the bisimulation as well and we have to check that they can mimic each others moves.

The entire bisimulation relation only contains five pairs, three are the in-situ transformation rules of Fig. 6 and two additional ones are needed. However, it is necessary to derive a large number of labels to prove that it is a bisimulation.

6 Discussion and Evaluation

We will now discuss the advantages and disadvantages of the two strategies and try to draw a conclusion about the state of the art in the field.

TGG approach. The direct bisimulation proof based on triple graph grammars uses little additional theory and can be carried out by resorting to standard proof methodology. Because of that it is more flexible than the borrowed context technique and can deal with the rules of the operational semantics without modification. On the other hand the setting is fairly abstract: there are no concrete graphs to work with and some creativity is required to define the bisimulation relation. Mechanization seems more difficult, but could be achievable using a theorem prover or a prover specializing in nested graph conditions [21].

Borrowed contexts. The borrowed context technique seems to be easier to mechanize: the label derivation process can be done fully automatically and, at least in the case where a finite bisimulation up-to context exists, there is some hope to find it via an algorithm as suggested in [3,9]. On the other hand it was necessary to adapt the operational rules in order to be able to apply the technique and we had to find an appropriate mixed semantics. We have some initial ideas for automatically generating the mixed semantics (by applying the transformation rules to the left-hand sides of the operational rules), but this is still an open problem. Furthermore we currently have to prove be-

havioural equivalence under all, also nonsensical, contexts. Restricting the number of contexts is also work in progress.

Summary. Although we were able to make both proofs work with a reasonable effort, it not clear whether the approaches scale. We conclude that additional techniques, in particular mechanisation, will be needed to address realistic languages such as the ones in [5]. Furthermore, both approaches required an a priori idea of how the correspondence between the models works. In both cases we found (weak) bisimilarity to be a suitable behavioural equivalence: it might be finer than other equivalences, but is easy to handle in proofs and it implies trace and failures equivalence.

In the future it will also be interesting to study refactoring cases. They promise to be easier since no transformation between distinct languages is performed. On the other hand, since refactorings often introduce optimization, it is less certain whether there will be a clear correspondence between source and target models such as the one exploited in our proofs.

Related work. The work closest to ours in its objective of showing semantic preservation for a transformation between models of different types is [6]. They present a mechanised proof of semantics preservation (wrt. some version of bisimilarity — the paper does not contain an explicit definition) for a transformation of automata to PLC-code, based on TGG rules. This proof faced some problems since it was not trivial to present graph transformation within Isabelle/HOL.

There are many papers which encode one specific formalism into another and then show full abstraction (with respect to some behavioural equivalence). In a sense, our work follows this tradition, but our idea was to choose a deliberately simple case study for which the arising problems can be studied in detail.

As opposed to general model transformation, there has been more work on showing correctness of refactorings. The methods presented in [29,22,18,7] address behaviour preservation in model refactoring, but are in general limited to checking a certain number of models. The employment of a congruence result is also proposed in [1] which uses the process algebra CSP as a semantic domain. The techniques used in [16,25] mainly treat state-based models, using set theory and predicate logic to show equivalences. A number of approaches also focus on preserving specific aspects instead of the full semantics (see [17]).

Instead of generally proving correctness of a transformation, a number of approaches, also in the area of compiler validation, instead carry out run-time checks of equivalence between given source and generated target model [18,19,14].

References

1. Bisztray, D., Heckel, R., Ehrig, H.: Verification of architectural refactorings by rule extraction. In: FASE '08. Volume 4961 of LNCS., Springer (2008) 347–361
2. Bonchi, F., König, B., Montanari, U.: Saturated semantics for reactive systems. In: Proc. of LICS '06, IEEE (2006) 69–80
3. Bonchi, F., Montanari, U.: Minimization algorithm for symbolic bisimilarity. In: ESOP '09. Volume 5502 of LNCS., Springer (2009) 267–284
4. Ehrig, H., König, B.: Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. MSCS **16**(6) (2006) 1133–1163

5. Engels, G., Kleppe, A., Rensink, A., Semenyak, M., Soltenborn, C., Wehrheim, H.: From UML Activities to TAAL - Towards Behaviour-Preserving Model Transformations. In: ECMDA-FA '08. Volume 5095 of LNCS., Springer (2008) 95–109
6. Giese, H., Glesner, S., Leitner, J., Schäfer, W., Wagner, R.: Towards verified model transformations. In: Workshop on Model Development, Validation and Verification. (2006) 78–93
7. Gorp, P.V., Stenten, H., Mens, T., Demeyer, S.: Towards automating source-consistent UML refactorings. In: UML 2003. Volume 2863 of LNCS., Springer (2003) 144–158
8. Hausmann, J.: Dynamic Meta Modeling: A Semantics Description Technique for Visual Modeling Languages. PhD thesis, University of Paderborn (2005)
9. Hirschkoff, D.: On the benefits of using the up-to techniques for bisimulation verification. In: TACAS '99. Volume 1579 of LNCS. (1999) 285–299
10. Hülsbusch, M., König, B., Rensink, A., Semenyak, M., Soltenborn, C., Wehrheim, H.: Showing full semantics preservation in model transformation – a comparison of techniques. Available from `http://www.ti.inf.uni-due.de/people/koenig/download/vmt-ext.pdf` (extended version).
11. Jensen, O.H.: Mobile Processes in Bigraphs. PhD thesis, University of Cambridge (2006)
12. Kastenberg, H., Kleppe, A., Rensink, A.: Defining object-oriented execution semantics using graph transformations. In: FMOODS '06. Volume 4037 of LNCS., Springer (2006) 186–201
13. Königs, A.: Model transformation with triple graph grammars. In: Workshop on Model Transformations in Practice. (2005)
14. Küster, J., Gschwind, T., Zimmermann, O.: Incremental development of model transformation chains using automated testing. In: MoDELS. Volume 5795 of LNCS. (2009) 733–747
15. Leifer, J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: CONCUR '00. LNCS, Springer (2000) 243–258
16. McComb, T., Smith, G.: Architectural Design in Object-Z. In: ASWEC'04, IEEE (2004) 77–86
17. Mens, T., Tourwé, T.: A survey of software refactoring. IEEE Trans. Software Eng. **30**(2) (2004) 126–139
18. Narayanan, A., Karsai, G.: Towards verifying model transformations. In: GT-VMT '06. Volume 211 of ENTCS. (2006) 185–194
19. Necula, G.: Translation validation for an optimizing compiler. In: PLDI '00. Volume 35 of SIGPlan Notices., ACM (2000) 83–95
20. Object Management Group: OMG Unified Modeling Language (OMG UML) – Superstructure, Version 2.2. `http://www.omg.org/docs/formal/09-02-02.pdf` (2009)
21. Pennemann, K.H.: Resolution-like theorem proving for high-level conditions. In: ICGT '08. Volume 5214 of LNCS., Springer (2008) 289–304
22. Pérez, J., Crespo, Y.: Exploring a method to detect behaviour-preserving evolution using graph transformation. In: Third International ERCIM Workshop on Software Evolution. (2007) 114–122
23. Rangel, G., König, B., Ehrig, H.: Deriving bisimulation congruences in the presence of negative application conditions. In: FOSSACS '08. Volume 4962 of LNCS., Springer (2008) 413–427
24. Rangel, G., Lambers, L., König, B., Ehrig, H., Baldan, P.: Behavior preservation in model refactoring using DPO transformations with borrowed contexts. In: ICGT '08. Volume 5214 of LNCS., Springer (2008) 242–256
25. Ruhroth, T., Wehrheim, H.: Refactoring object-oriented specifications with data and processes. In: FMOODS '07. Volume 4468 of LNCS., Springer (2007) 236–251
26. Sassone, V., Sobociński, P.: Reactive systems over cospans. In: LICS '05, IEEE (2005) 311–320
27. Schürr, A., Klar, F.: 15 years of triple graph grammars. In: ICGT '08. Volume 5214 of LNCS., Springer (2008) 411–425

28. van Glabbeek, R.: The linear time - branching time spectrum II. In: CONCUR '93. Volume 715 of LNCS., Springer (1993) 66–81
29. van Kempen, M., Chaudron, M., Kourie, D., Boake, A.: Towards proving preservation of behaviour of refactoring of UML models. In: SAICSIT '05. (2005) 252–259

A Details for Proof Strategy 1

This appendix gives a more complete picture of the proof sketched in Section 4. We start with a formalisation of our first observation that the static structure of graphs stays the same (except for start edges) when semantic rules are applied.

Proposition 13. *Let $G_A \in \mathcal{G}_A^r$ be an $\mathcal{A}$-graph. If $G_A \xrightarrow{r} G'_A$ for some $r \in R_A$ then $\pi_{\mathsf{T}^s_\mathsf{A} \setminus \mathsf{start}}(G_A) = \pi_{\mathsf{T}^s_\mathsf{A} \setminus \mathsf{start}}(G'_A)$, where $\mathsf{T} \setminus \mathsf{start}$ is the type T without the* start*-edge. A corresponding property holds for $\mathcal{B}$.*

This means that the correspondence structure of $\mathcal{A}$ and $\mathcal{B}$-graphs is kept. The following propositions illustrate some correspondences which can be shown by induction on the application of TGG-rules. The first concerns start-edges:

Proposition 14. *Let $G \in \mathcal{G}_{AB}$, v_S an* S*-node, v_A an* A*-node and let $corr(v_S, v_A)$. Then*

$$v_S \textit{ has a } \mathsf{start}\textit{-edge} \quad \textit{iff} \quad v_A \textit{ has a } \mathsf{start}\textit{-edge}.$$

Moverover, there is exactly one start-edge on the $\mathcal{A}$- and one on the $\mathcal{B}$-side. The next correspondence properties hold between next-edges and connections via C-nodes.

Proposition 15. *Let $G \in \mathcal{G}_{AB}$, v_S an* S*-node, v_A an* A*-node and let $corr(v_S, v_A)$.*

- *If there is a* C*-node v_C, such that $\mathsf{conn}(v_A, v_C)$, then there is an* S*-node v'_S and an* A*-node v'_A such that $\mathsf{next}(v_S, v'_S)$, $\mathsf{act}(v_C, v'_A)$ and $corr(v'_S, v'_A)$.*
- *If there is an* S*-node v'_S such that $\mathsf{next}(v_S, v'_S)$, then there is a* C*-node v_C and an* A*-node v_A such that $\mathsf{conn}(v_A, v_C)$, $\mathsf{act}(v_C, v'_A)$ and $corr(v'_S, v'_A)$.*

Together with the uniqueness of predicate $corr$ (Prop. 10, exactly one S-node related to one A-node), these propositions are essential for showing that the relation $\mathcal{R}$ given in Section 4 indeed defines a weak bisimulation.

Proof. of Theorem 11. Taking the relation $\mathcal{R}$, we need to show the property of mutual simulation. We start with the requirement of initial states being in the relation. The initial states of the LTSs are G_A^0 and G_B^0 and they satisfy the conditions of $\mathcal{R}$ since they are directly generated by projection from the combined graph (condition (1)), Prop. 14 guarantees (2) and they have no run-time elements such as tokens, offers or program counters, so condition (3) is trivially satisfied, and (4) follows from the TGG rules.

Now assume $(G_A, G_B) \in \mathcal{R}$ and $G_A \xrightarrow{r_1} G'_A$. As we are looking at the LTSs with labels renamed according to map_A and map_B, r_1 (the label of the transition) in principle is either init, move or τ. We need to show that there is some G'_B such that $G_B \xRightarrow{\widehat{r_1}} G'_B$ with $(G'_A, G'_B) \in \mathcal{R}$. However, as we are interested in the particular semantic rule applied during the step, we will instead directly look at the original LTSs and show that map_A and map_B map rule names to the same label.

$r_1 = \mathsf{initA}$**:** Let $\langle L_1, I_1, R_1, \mathcal{N}_1\rangle$ be the rule for initA in Fig. 3. If r_1 is applicable in G_A, we have a match $m_1 : L_1 \to G_A$, i.e., a node v_S such that $\mathsf{start}(v_S)$. From this we construct a match $m_2 : L_2 \to G_B$ for the rule $r_2 = \mathsf{initB}$ (both being mapped to init by map_A and map_B) being defined as $\langle L_2, I_2, R_2, \mathcal{N}_2\rangle$. The match m_2 maps the A-node in L_2 to the due to Prop. 10 uniquely existing A-node v_A in G_B such that $corr(v_S, v_A)$. By condition (2) of $\mathcal{R}$ we get $\mathsf{start}(v_A)$. Thus r_2 is applicable in G_B. Once the rules are applied, we have a graph G'_A with one new T-node v_T with $\mathsf{pc}(v_T, v_S)$ minus the (only) start-edge $\mathsf{start}(v_S)$, and a similar construction for G_B. The pair (G'_A, G'_B) is in $\mathcal{R}$ since (1) the static structure without start edges is kept (Prop. 13); the pair (v_S, v_A) satisfies (2) since both start-edges are deleted, all other pairs satisfy (2) since they are unchanged; (3) is met because we have $\exists v_T : \mathsf{pc}(v_T, v_S) \wedge \mathsf{token}(v_A)$ and no offers are created, and since by (4) we know that no offers have been existing before; and (4) is met since the two start-edges have been deleted and for them exactly one pc- and one token-edge has been created (and no offers).

$r_1 = \mathsf{movePC}$**:** Since r_1 is applicable in G_A, we have nodes v_S, v'_S, v_T in G_A s.t. $\mathsf{pc}(v_T, v_S) \wedge \mathsf{next}(v_S, v'_S)$. By (1) and Prop. 10 there are unique nodes v_A, v'_A in G_B s.t. $corr(v_S, v_A)$ and $corr(v'_S, v'_A)$. By (1) and Prop. 15 there exists v_C s.t. $\mathsf{conn}(v_A, v_C) \wedge \mathsf{act}(v_C, v'_A)$. By (3) there are now two possible cases:

1. $\mathsf{token}(v_A) \wedge \forall v_C$ s.t. $\mathsf{conn}(v_A, v_c) : \neg\mathsf{offer}(v_C)$.
   Thus rule $r_2 = \mathsf{createO}$ matches on v_A and v_C (and both r_1 and r_2 are mapped to move). In the resulting graph G'_A the pc-edge from v_T to v_S has been deleted and one from v_T to v'_S created. G'_B has a new offer-edge on v_C. $(G'_A, G'_B) \in \mathcal{R}$ since (1) static structure is kept, (2) no start edges are touched, (3) both pairs (v_S, v_A) and (v'_S, v'_A) satisfy the condition, the others are unchanged, and (4) since no start edges are created and the new offers sits on a node following node possessing a token.
2. $\neg\mathsf{token}(v_A) \wedge \exists v_C, v'_A : \mathsf{token}(v'_A) \wedge \mathsf{offer}(v_C) \wedge \mathsf{conn}(v'_A, v_C) \wedge \mathsf{act}(v_C, v_A)$. Then the invisible rule moveT (being mapped to τ) is applicable in G_B leading to a graph G''_B in which $\mathsf{token}(v_A)$ holds. Moreover, by (4) and rule moveT we know that for all v'_C s.t. $\mathsf{conn}(v_A, v'_C)$ we have $\neg\mathsf{offer}(v'_C)$. Now we reached the first case again and proceed like that. In summary, we get in the renamed LTS
   $$G_B \xrightarrow{\tau} G''_B \xrightarrow{\mathsf{move}} G'_B, \text{ i.e. } G_B \xRightarrow{\widehat{\mathsf{move}}} G'_B$$
   and furthermore $(G'_A, G'_B) \in \mathcal{R}$.

Reverse direction: assume $G_B \xrightarrow{r_2} G'_B$. We need to show that there is some G'_A such that $G_A \xRightarrow{\widehat{r_2}} G'_A$ and $(G'_A, G'_B) \in \mathcal{R}$. Again, we argue on the level of LTSs before renaming.

$r_2 = \mathsf{initB}$**:** Similar to initA.

$r_2 = \mathsf{createO}$**:** Since r_2 is applicable in G_B there are nodes $v_A, v_C : \mathsf{token}(v_A) \wedge \mathsf{conn}(v_A, v_C) \wedge \forall v'_C$ s.t. $\mathsf{conn}(v_A, v'_C) : \neg\mathsf{offer}(v_C)$. By (1) and Prop. 10 there is a unique node v_S s.t. $corr(v_S, v_A)$. By (3) $\exists v_T : \mathsf{pc}(v_T, v_S)$. By (1) and Prop. 15 $\exists v'_A, v'_S$ s.t. $\mathsf{next}(v_S, v'_S) \wedge \mathsf{act}(v_C, v'_A) \wedge corr(v'_S, v'_A)$. Hence rule

movePC (mapped to move like createO) is applicable in G_A. The rest follows from a reasoning similar to case movePC.

$r_2 = \mathsf{moveT}$**:** In this case, we have an invisible step on the $\mathcal{B}$-side. If r_2 is applicable in G_B, then $\exists v_A, v_C, v'_A : \mathsf{token}(v_A) \wedge \mathsf{offer}(v_c) \wedge \mathsf{conn}(v_A, v_C) \wedge \mathsf{act}(v_C, v'_A)$. By (1) and Prop. 10 $\exists v_S, v'_S : corr(v_S, v_A) \wedge corr(v'_S, v'_A)$. By Prop. 10 and Prop. 15 we get $\mathsf{next}(v_S, v'_S)$. By (4) we get $\neg\mathsf{token}(v'_A)$. By (3) we have $\exists v_T : \mathsf{pc}(v_T, v'_S)$ and thus by (4) $\neg\exists v_T : \mathsf{pc}(v_T, v_S)$. Applying rule r_2 leads to a graph G'_B in which $\mathsf{token}(v'_A)$ and $\neg\mathsf{offer}(v_C) \wedge \neg\mathsf{token}(v_A)$ holds. Because of (4) (the only possible offer was on v'_C) we know that $\forall v'_C$ s.t. $\mathsf{conn}(v'_A, v'_C) : \neg\mathsf{offer}(v'_C)$. The pair (G_A, G'_B) is thus in $\mathcal{R}$ and furthermore $G_A \stackrel{\hat{\tau}}{\Longrightarrow} G_A$ which completes the proof.

□

B Details for Proof Strategy 2

B.1 Summary of the Borrowed Context Technique

We will first discuss the main issues for the borrowed context technique underlying proof strategy 2. Afterwards we will give the formal definitions and prove the congruence theorem. A similar proof was already given in [23], but it has to be redone, since we are working in a different setting, including weak and saturated moves. For label derivation techniques and weak bisimilarity see also [11].

- *DPO rules:* we use rules as described in Definition Def. 5, but represent them as pairs of morphisms $L \leftarrow I \rightarrow R$, which describe the embedding of the interface into the left-hand and right-hand side. Furthermore negative application conditions are given by a morphism $L \rightarrow N$ (or alternatively $I \rightarrow N$ if we omit the parts of N that are already contained in L). Rule application is then performed by doing a categorical double-pushout construction.
- *Negative application conditions:* since the rules used in the case study feature negative application conditions, we have to incorporate negative borrowed contexts. This has already been studied in [23]. For this paper this means that we do not only consider *positive* borrowed contexts $J \rightarrow F \leftarrow K$, but additionally *negative* borrowed contexts of the form $K \rightarrow N \leftarrow N$ that detail what must *not* be present in order to perform the step.
- *Saturated bisimilarity:* most bisimulation games are played in such a way that labels have to match exactly. However, it turns out that requiring that the minimal borrowed contexts are the same in both cases does not give us the coarsest possible congruence. Hence bisimulation is defined in the following way: every move with a minimal context, i.e., everything that is borrowed is necessary to complete the left-hand side, can be answered by borrowing more than what is actually needed. Similarly the negative borrowed context of the answering move might be more permissive.
- *Weak bisimilarity:* the ideas behind saturated bisimilarity then simplify to a certain extent the integration of weak transitions (see also Section 2.5). A strong transition, borrowing a (minimal) context $J \rightarrow F \leftarrow K$ can be answered by borrowing the

same context and using this context for completing the left-hand sides of several consecutive transitions, leading to a weak transition. Negative borrowed contexts are then derived with respect to the part of F that is still left.

We will in the following prove that if the bisimulation game is played according to the restrictions explained above, then bisimilarity is a congruence. That is, if we determine that two graphs with interfaces $(J \to G), (J \to G')$ are bisimilar, then they are also bisimilar if we extend them with a context $J \to E \leftarrow K$ and view them as extended graphs with a new interface K.

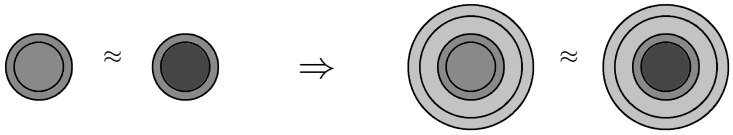


Fig. 11. Schematic representation of the congruence result

B.2 Theoretical Background

We will now show the relevant congruence theorem for proof strategy 2, i.e., for label derivation via borrowed context. The proof is an extension of a proof from [23] to the saturated and weak case. We will work in a categorical setting and specifically we need the notions of pushout and pullback.

We first introduce the notion of DPO rewriting, working in the category Graph. Note that for the purposes of this paper we consider only injective morphisms for rules and matches.

Definition 16 (DPO rewriting). *A* DPO rule *is of the form $r = \langle L \xleftarrow{\ell} I \xrightarrow{r} R, \mathcal{N} \rangle$ where $\mathcal{N}$ is a set of negative application conditions of the form $n_i \colon L \to N_i$. All morphisms ℓ, r, n_i must be injective.*

An injective match $m \colon L \to G$ satisfies a negative application condition $n_i \colon L \to N_i$ on L if and only if there is no injective morphism $q \colon N \to G$ with $q \circ n = m$. (See diagram below on the left.)

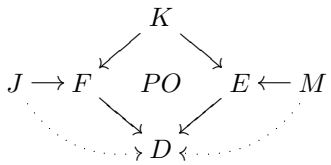
$$\begin{array}{ccc} N_i & \xleftarrow{n_i} & L \\ & \searrow^{q} & \downarrow{\scriptstyle m} \\ & & G \end{array} \qquad\qquad \begin{array}{ccccc} N_i & \longleftarrow & L & \xleftarrow{\ell} & I & \xrightarrow{r} & R \\ & & {\scriptstyle m}\downarrow & PO & \downarrow & PO & \downarrow \\ & & G & \longleftarrow & C & \longrightarrow & H \end{array}$$

If the negative application condition is satisfied, the application *of r to G for a match $m \colon L \to G$ is performed via the diagram consisting of two pushouts above on the right (where all morphisms are injective) and results in the graph H.*

The notion of rewriting introduced here is the categorical counterpart to the more concrete set-based notion defined in Section 2.3.

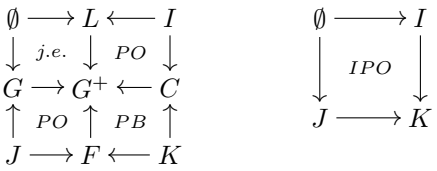
We will now review the notion of contexts, also called cospans.

Definition 17 (context, cospan). *A* context *(also called* cospan*) consists of two injective graph morphisms $J \to F \leftarrow K$. The composition of a two cospans is performed via taking the pushout.*



In order to define label derivation via borrowed contexts we need the notion of IPO-like squares. Formally they are groupoidal idem pushouts in the bicategory of cospans [26]. For the purposes of this paper and the congruence proof which will follow, it is unnecessary to delve into this complex theory. We will only define the notion of IPO-like squares and state some of its properties.

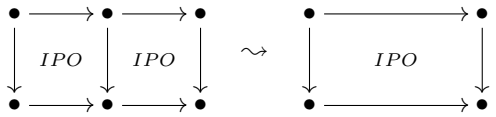
Definition 18 (IPO-like square). *An* IPO-like square *is a commuting diagram in the category of cospans which has the form shown on the left below. Note that "j.e." signifies that the two arrows $L \to G^+$, $G \to G^+$ must be jointly epi.*



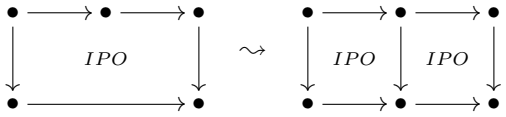
We will in the following work in the category of cospans, which means that we draw the diagram as a commuting diagram in the category of cospans as shown on the right above.

Proposition 19 (Properties of IPO-like squares). *IPO-like squares enjoy the following properties:*

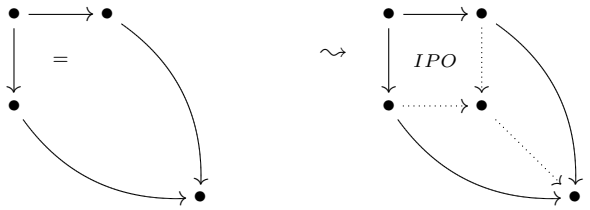
(A) If the two inner squares below are IPO-like, then the outer square is IPO-like.



(B) Every IPO-like square splits (non-uniquely) into two IPO-like squares.



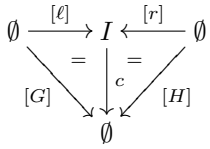
(C) Every commuting square contains an IPO-like squares as shown in the diagram below.



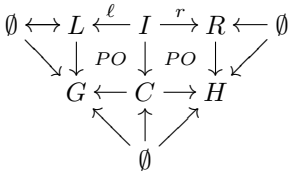
Proof. This is a corollary of related results in [26].

In order to change the whole presentation to cospans, which gives us a more abstract setting which is easier to handle, we also have to define DPO rewriting via cospans. In order to do this we consider every rule span $L \xleftarrow{\ell} I \xrightarrow{r} R$ as a pair of cospans $[\ell]\colon \emptyset \to L \xleftarrow{\ell} I$, $[r]\colon \emptyset \to R \xleftarrow{r} I$.

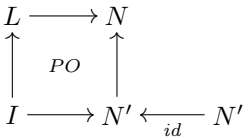
Now view two graphs G, H as cospans $[G] = \emptyset \to G \leftarrow \emptyset$, $[H] = \emptyset \to H \leftarrow \emptyset$. Then we can give an alternative definition of DPO rewriting, stating that G rewrites to H via the rule given above if and only if there exists a cospan c such that the diagram below commutes.



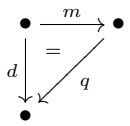
Note that the diagram on the right above, if spelled out in the base category, looks as follows. That is, it is equivalent to DPO rewriting as in Def. 16.



In the following we will also regard negative application conditions or negative borrowed contexts as cospans. Every negative application condition $L \xrightarrow{n} N$ is converted into a cospan $m\colon I \to N' \xleftarrow{id} N'$ by taking a pushout complement as follows. (Note that this might lead to zero or more cospans m.)



A cospan d satisfies a cospan m seen as a negative application condition (in symbols $d \models m$) if there exists no cospan q making the diagram below commute.



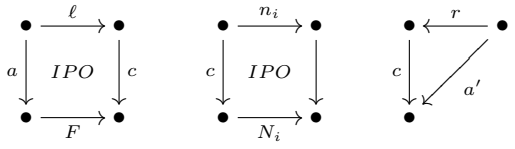
For a DPO rewriting step we derive from $\mathcal{N}$ (the set of negative application conditions) all cospans m as described above, obtaining a set $\mathcal{M}$ of cospans. Then we check whether the cospan c in the (rewriting) diagram above satisfies all negative application conditions in $\mathcal{M}$ (in symbols $c \models \mathcal{M}$). This is equivalent to the condition given in Def. 16.

We say that a cospan m implies another cospan m' ($m \models m'$) if for every cospan c, $c \models m$ implies $c \models m'$. Analogously we define $\mathcal{M} \models \mathcal{M}'$ for sets of cospans.

Now strong and weak transitions are derived as follows.

Definition 20 (strong/weak transitions). *Let $a\colon \emptyset \to J$ be a graph with interface, let $\ell, r\colon \emptyset \to I$ be a rule, called p, and let $\mathcal{M}$ be a set of negative application conditions of the form $n_i\colon I \to N_i$ (all arrows are cospans).*

Now let F, c be cospans such that the square below on the left is an IPO-like square and take the set $\mathcal{N}$ of all arrows N_i that form an IPO-like square with c and some cospan $n_i \in \mathcal{M}$.



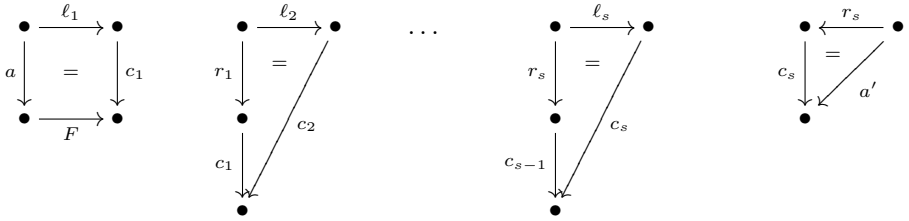
Now let $a' = r; c$ be constructed as shown above. Then there exists a strong transition *from a to a' with labels $map(p), F, \mathcal{N}$ and we write*

$$a \xrightarrow{map(p),F,\mathcal{N}} a'$$

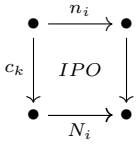
Let again $a\colon \emptyset \to J$ be a graph with interface, let $\ell_k, r_k\colon \emptyset \to I_k$ be rules, called p_k with $1 \leq k \leq s$, and let $\mathcal{N}_k$ be the corresponding sets of negative application conditions. We obtain a weak transition

$$a \overset{\alpha,F,\mathcal{N}}{\Longrightarrow} a'$$

if we have the following situation:



and $\mathcal{N}$ consists of all arrows N_i that form an IPO-like square with c_k and some cospan $n_i \in \mathcal{N}_k$ (see square below).



Furthermore there is at most one visible rule p_k and we set $\alpha = map(p_k)$. If there is no transition or no visible transition we have $\alpha = \epsilon$.

Hence in the weak move we borrow a context F that is large enough to complete all left-hand sides ℓ_k and which might even contain elements that are not required in order to reduce. That is we follow the ideas of saturated semantics [2].

Definition 21 (weak bisimulation). *Assume a given set of rewriting rules with negative application conditions. A* weak bisimulation *R is a symmetric relation on cospans of the form $\emptyset \to J$ where two cospans a, b with $a\,R\,b$ have the same target object. Furthermore for every strong transition*

$$a \overset{\alpha,F,\mathcal{N}}{\longrightarrow} a'$$

there exist weak transitions

$$b \overset{\hat{\alpha},F,\mathcal{N}_i}{\Longrightarrow} b'_i$$

for $1 \leq i \leq t$ such that

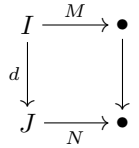
- *$a'\,R\,b'_i$ for all $1 \leq i \leq t$*
- *$\mathcal{N} \models \mathcal{N}_1 \vee \cdots \vee \mathcal{N}_t$ (this means that for every cospan c with $c \models \mathcal{N}$ there exists an index $i \in \{1, \ldots, t\}$ with $c \models \mathcal{N}_i$)*

Weak bisimilarity *(denoted by $\approx$) is the largest weak bisimulation.*

In the case study treated in this paper it will always be the case that b can answer with a single move.

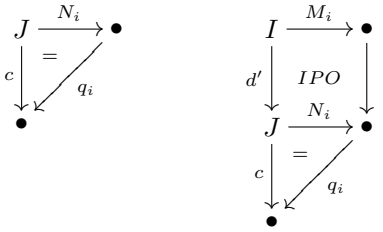
Now we prove the main theorem which is crucial to show behaviour preservation of the model transformation rules: weak bisimilarity is a congruence. First we need the following lemma:

Lemma 22. *Let $\mathcal{M}, \mathcal{M}_1, \ldots, \mathcal{M}_t$ be sets of negative application conditions of the form $I \to \bullet$ for a fixed object I. Furthermore assume a given cospan $d\colon I \to J$ and construct the set $\mathcal{N}$ (respectively $\mathcal{N}_1, \ldots, \mathcal{N}_t$) which consists of all arrows N which are obtained by deriving IPO-like squares with arrows d and $M \in \mathcal{M}$ (respectively $M \in \mathcal{M}_1, \ldots, \mathcal{M}_t$) as shown below.*

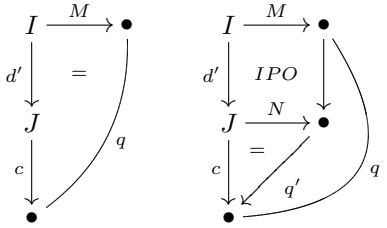


Then $\mathcal{M} \models \mathcal{M}_1 \vee \cdots \vee \mathcal{M}_t$ implies $\mathcal{N} \models \mathcal{N}_1 \vee \cdots \vee \mathcal{N}_t$.

Proof. Assume, by contradiction, that $\mathcal{M} \models \mathcal{M}_1 \vee \cdots \vee \mathcal{M}_t$ but $\mathcal{N} \not\models \mathcal{N}_1 \vee \cdots \vee \mathcal{N}_t$. Hence there exists a cospan c with $c \models \mathcal{N}$ and $c \not\models \mathcal{N}_i$ for all $1 \leq i \leq t$. This implies the existence of negative conditions $N_i \in \mathcal{N}_i$ and cospans q_i such that the diagrams below on the left all commute. Each N_i was derived from some M_i by taking an IPO-like square with d'. Hence we can attach those squares and obtain the commuting diagrams below on the right.



This implies that $d'; c \not\models M_i$ for some $M_i \in \mathcal{M}_i$. Therefore $d'; c \not\models \mathcal{M}_i$. To obtain a contradiction it is now sufficient to show $d'; c \models \mathcal{M}$. Assume that $d'; c \not\models \mathcal{M}$. Then there would exist an $M \in \mathcal{M}$ and a cospan q such that the diagram below on the left commutes. Then, due to Property (C) of Proposition 19 we can find an IPO-like square of d', M_i as shown on the right.



However $N \in \mathcal{N}$ which implies $c \not\models \mathcal{N}$, which is a contradiction. □

Theorem 23 (Weak bisimilarity is a congruence). *Assume a given set of rewriting rules with negative application conditions. Let $a, b\colon \emptyset \to J$ be two cospans such that $a \approx b$ (with respect to the given rules). Then for any cospan $d\colon J \to K$ it holds that $a; d \approx b; d$.*

Proof. We assume a given weak bisimulation R and define

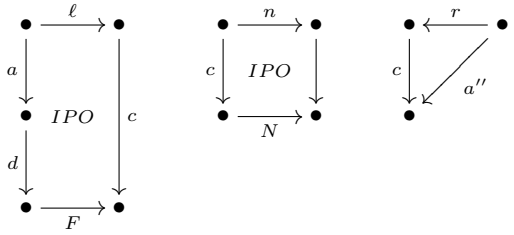
$$\hat{R} = \{(a; d, b; d) \mid a \, R \, b, d \text{ arbitrary}\}.$$

It is enough to show that $\hat{R}$ is a weak bisimulation. So take a pair $a; d \, \hat{R} \, b; d$ and assume that

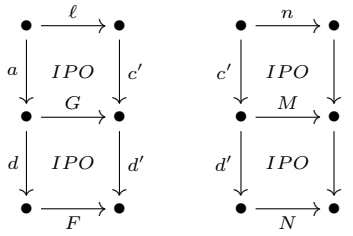
$$a; d \xrightarrow{\alpha, F, \mathcal{N}} a''$$

with a rule p consisting of ℓ, r where $map(p) = \alpha$. Hence, by Definition Def. 20 we have the following situation where $\mathcal{N}$ consists of all negative contexts N that can be

derived from c and the negative application conditions n_i of the rule.



Now by Property (B) of Proposition 19 we know that the IPO-like squares split (possibly in several ways) into IPO-like squares as shown below (where $c = c'; d'$).



For the left square we choose one such split and for the right square we take all possible splits, obtaining a set $\tilde{\mathcal{M}}$ of negative contexts M. Now we have that

$$a \overset{\alpha, G, \mathcal{M}}{\longrightarrow} a'$$

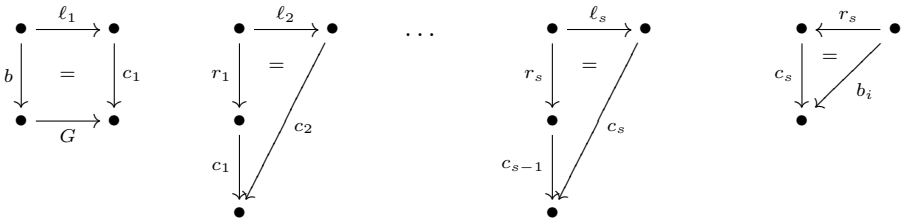
for $a' = r; c'$ (and so $a'; d' = r; c'; d' = r; c = a''$). Furthermore $\tilde{\mathcal{M}} \subseteq \mathcal{M}$ and the $M \in \mathcal{M} \backslash \tilde{\mathcal{M}}$ are exactly the negative conditions in $\mathcal{M}$ that do not have an IPO-like square with d' (otherwise they would be contained in $\tilde{\mathcal{M}}$). Hence $\mathcal{N}$ can be obtained from $\mathcal{M}$ by deriving all IPO-like squares with d'.

Now since $a \, R \, b$ we conclude, by the definition of bisimulation, that there exist weak transitions

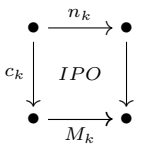
$$b \overset{\hat{\alpha}, G, \mathcal{M}_i}{\Longrightarrow} b_i'$$

for $1 \leq i \leq t$ such that $a' \, R \, b_i'$ for all $1 \leq i \leq t$ and $\mathcal{M} \models \mathcal{M}_1 \vee \cdots \vee \mathcal{M}_t$.

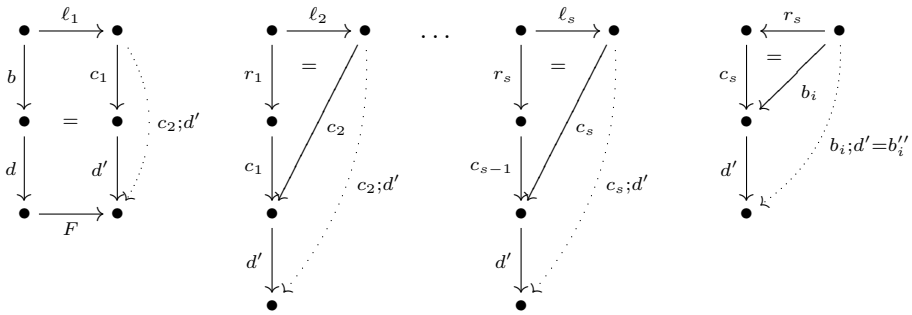
Hence there are rules $p_1, \ldots, p_k$ ($1 \leq k \leq s$) consisting of cospans ℓ_k, r_k and negative application conditions and we have the following commuting diagrams in the cospan category.



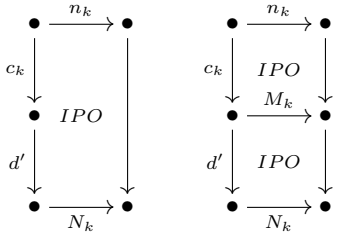
Furthermore $\mathcal{M}_i$ is obtained by taking all arrows M_k that form an IPO-like square with c_k and some negative application condition of the k-th rule.



Now extend the IPO-like square consisting of b, G, ℓ_1, c_1 with the IPO-like square consisting of d, F, G, d' and obtain the commuting square shown below on the left. Furthermore add d' to the triangles and obtain the remaining commuting diagrams.



In addition let $\mathcal{N}_k$ be the set of negative borrowed contexts N_k that are obtained by forming an IPO-like square of $c_k; d$ and a negative application n_k condition of the k-the rule (see diagram on the left below). By taking all possible splits of this diagram we obtain diagrams of the form shown on the right. Hence note that $\mathcal{N}_k$ can be obtained by taking all negative conditions contained in $\mathcal{M}_k$ and taking IPO-like squares with d'.



We can now infer that

$$b' \overset{\hat{\alpha},F,\mathcal{N}_i}{\Longrightarrow} b_i''$$

where $b_i'' = b_i; d'$. Since $a'' = a'; d'$ we can infer that $a'' \,\hat{R}\, b_i''$.

It is left to show that $\mathcal{N} \models \mathcal{N}_1 \vee \cdots \vee \mathcal{N}_t$. We have that $\mathcal{M} \models \mathcal{M}_1 \vee \cdots \vee \mathcal{M}_t$ and that $\mathcal{N}$ $(\mathcal{N}_1, \ldots, \mathcal{N}_t)$ can be obtained from $\mathcal{M}$ $(\mathcal{M}_1, \ldots, \mathcal{M}_t)$ by taking all IPO-like squares with d'. Hence we apply Lemma 22 and get the desired result. □

Note also that, by an easy modification of the proof, the up-to-context technique described in [4] works also in this setting.

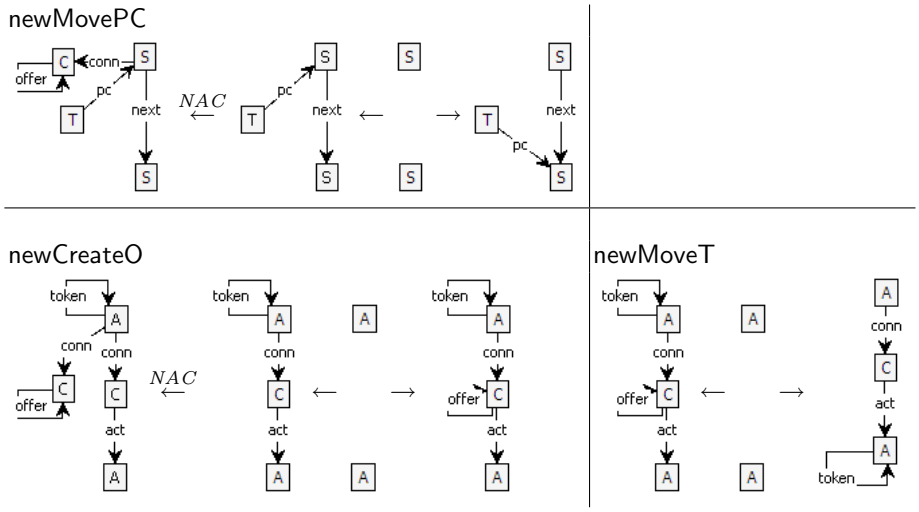
B.3 The Complete Mixed Semantics

In this section we present the modified rules of the operational semantics (originally given in Fig. 3) and the entire set of rules of the mixed semantics (explained and introduced in Section 5.3). Since our proof technique guarantees bisimulation in *every* context, not only in the ones that can be created via the given transformation, there are rules that seem to be useless, but are needed for the proof.

Modified operational rules. As explained in Section 5.3 we modify the rules movePC, createO and moveT of Fig. 3, which become newMovePC, newCreateO, newMoveT. Note that the rules initA and initB remain unchanged.

We apply the following two changes: we add NACs and we minimize the interface towards the environment by deleting and recreating connections. This mean also that rule newMoveT now contains two A-nodes (instead of just one for the case of moveT).

These new rules do not change the semantics for the following reasons: the negative application condition for the (source) rule newMovePC is uncritical since it only refers to elements of the target language. Second, the other NAC of rule createO is automatically enforced for newCreateO because of the dangling condition. Finally, the restrictions on language $\mathcal{B}$ guarantee that a C-node has exactly one A-predecessor and one A-successor.



Type graph of the mixed models. The in-situ rules allow no strict separation of source and target models. Therefore, the type graph of the mixed models (see Fig. 12) is a mixture of both original type graphs. Keep in mind that the labels within the nodes are hidden loops (see Section 2.1).

Movement of the pc*-edge (reachable graphs).* The following four rules show the movement of the pc-ege in mixed models, which are reachable via the in-situ transformation rules (although the NACs might never apply).

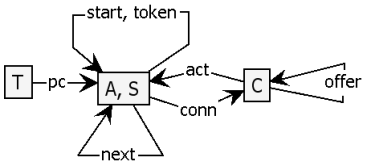
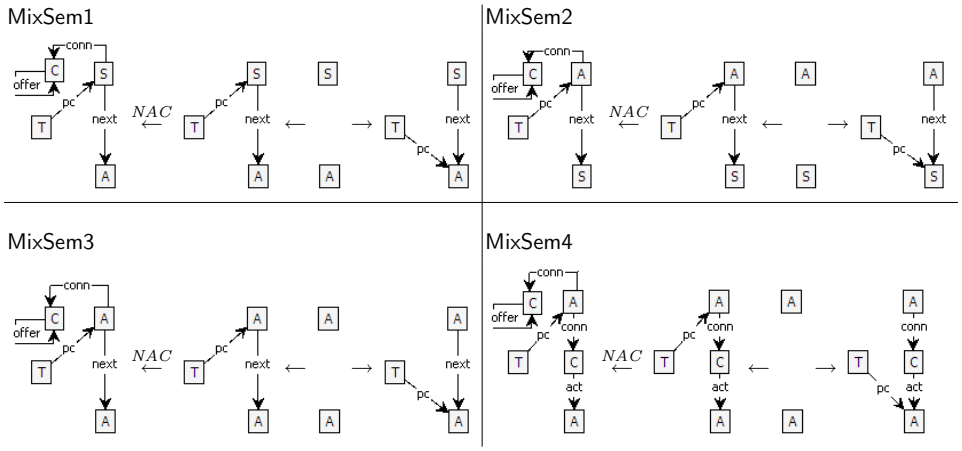
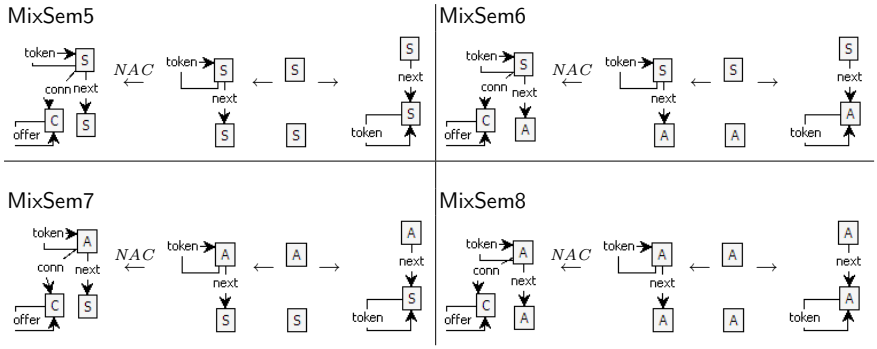


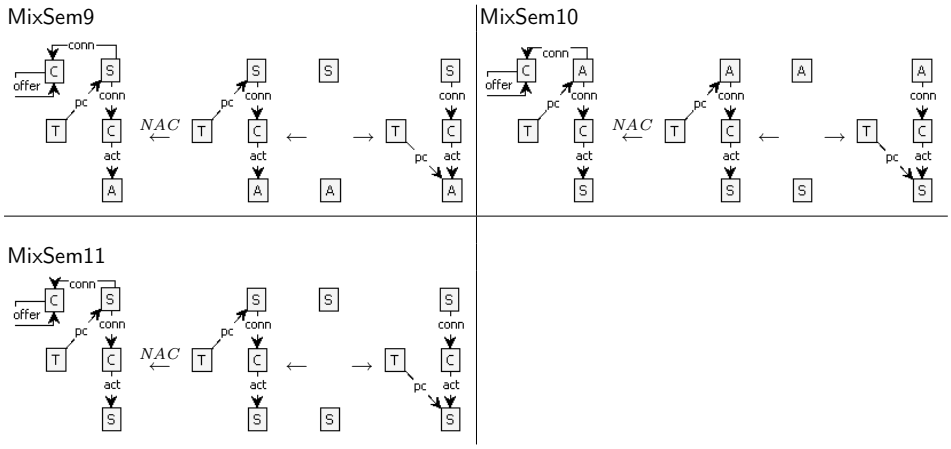
Fig. 12. Typegraph for the in-situ transformation



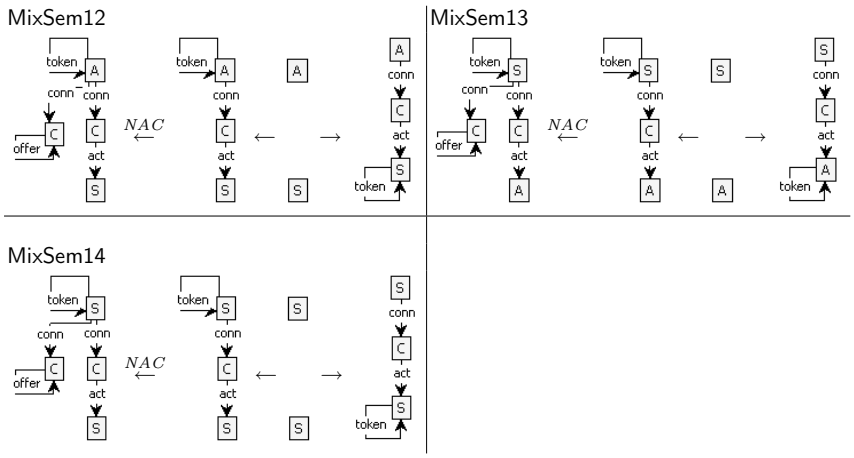
Movement of the token*-loop (reachable graphs).* The following four rules similarly describe the movement of the token-loop in mixed models, which are again reachable during model transformation.



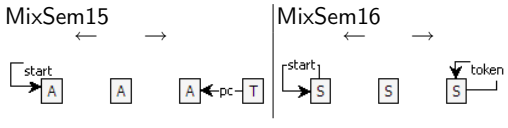
Movement of the pc*-edge (unreachable graphs).* The next three rules describe the movement of the pc-edge in mixed models, that are *not* reachable by the transformation rules, but which are needed to prove bisimilarity.



Movement of the token*-loop (unreachable graphs).* The following three rules show the movement of the token-loop in mixed models which are *not* reachable by the in-situ transformation rules, but again needed to prove bisimilarity.



Initialization rules. Finally, the last two rules are needed as a counterpart to the initialization rules.



We still need to give the mapping from the rule-names to the observable labels (see Section 2.5). The in-situ approach needs just one mapping and we use the following:

$$
\begin{array}{rrl}
map: & \mathsf{initA} & \mapsto \mathsf{init} \\
 & \mathsf{initB} & \mapsto \mathsf{init} \\
 & \mathsf{MixSem15} & \mapsto \mathsf{init} \\
 & \mathsf{MixSem16} & \mapsto \mathsf{init} \\
 & \mathsf{newMovePC} & \mapsto \mathsf{move} \\
 & \mathsf{newCreateO} & \mapsto \mathsf{move} \\
 & \mathsf{MixSem1} & \mapsto \mathsf{move} \\
 & \vdots & \vdots \\
 & \mathsf{MixSem14} & \mapsto \mathsf{move} \\
 & \mathsf{newMoveT} & \mapsto \tau
\end{array}
$$

There are no mixed counterparts to the rule newMoveT for the following reasons:

- The C-nodes are never in the interface, and therefore it is impossible to borrow something on an C-node (such as an single offer-edge). The borrowed context always provides the complete C-node, including all incident edges.
- The "offering behaviour" is not carried over into the mixed semantics. The mixed rules only allow direct movement of the token from the upper to the lower node. For this reason we know that after a newCreateO-step, the next step *must* be newMoveT (as answering move), which is invisible to the enviroment.

B.4 The Complete Bisimulation Relation and Excerpts from the Proof

The five pairs of graphs, given in Fig. 13, consitute the entire bisimulation relation, needed to show (weak) bisimilarity of the left-hand and right-hand sides of the in-situ transformation rules of Fig. 6 with respect to the rules of the mixed semantics (see Section B.3).

The lower row contains two new pairs, which are necessary to obtain a bisimulation. The left pair is added, since starting with the pair consisting of a single S-node and a single A-node, the A-node can perform an newCreateO-step, and the S-node has to answer (using rule MixSem13). This step is depicted in Fig. 10 (ignoring the weak newMoveT-step). The pair below on the left appears due to the moves depicted in Fig. 14.

Note that in most cases we do not obtain new pairs due to heavy use of the up-to-context technique. For instance, Fig. 15 shows an example where the application of two corresponding rules leads to a pair of graphs corresponding to the pair of graphs we started with (plus additional context, in this case a T-node and a pc-edge). The up-to-context proof technique says that in these cases we can stop if the pair is in the relation after removal of the additional context.

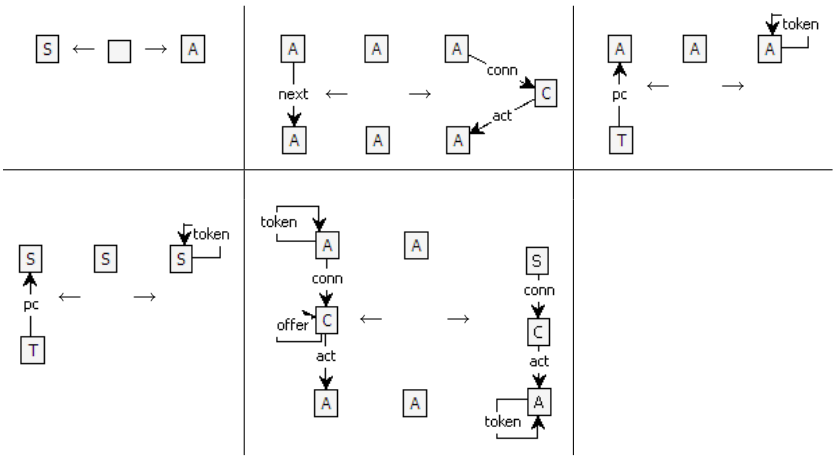


Fig. 13. The five pairs of graphs in the bisimulation relation

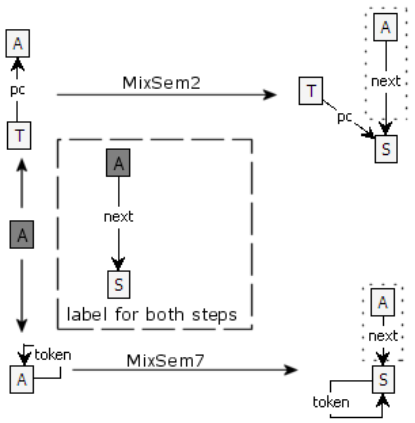


Fig. 14. Label derivation for the second in-situ transformation rule, resulting in a new pair of the bisimulation.

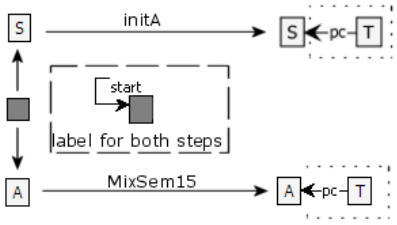


Fig. 15. Label derivation for the first in-situ transformation rule, resulting in a pair of graphs already in the bisimulation relation.