

A Temporal Graph Logic for Verification of Graph Transformation Systems*

Paolo Baldan¹, Andrea Corradini², Barbara König³, Alberto Lluch Lafuente²

¹ Dipartimento di Matematica Pura e Applicata, Università di Padova
baldan@math.unipd.it

² Dipartimento di Informatica, Università di Pisa
{andrea,llafuente}@di.unipi.it

³ Abt. für Informatik und Ang. Kognitionswissenschaft, Universität Duisburg-Essen
barbara.koenig@uni-due.de

Abstract. We extend our approach for verifying properties of graph transformation systems using suitable abstractions. In the original approach properties are specified as formulae of a propositional temporal logic whose atomic predicates are monadic second-order graph formulae. We generalize this aspect by considering more expressive logics, where edge quantifiers and temporal modalities can be interleaved, a feature which allows, e.g., to trace the history of objects in time. After characterizing fragments of the logic which can be safely checked on the approximations, we show how the verification of the logic over graph transformation systems can be reduced to the verification of a logic over suitably defined Petri nets.

1 Introduction

Graph Transformation Systems (GTSs) are suitable modeling formalisms for systems involving aspects such as object-orientation, concurrency, mobility and distribution. The use of GTSs for the verification and analysis of such systems is still at an early stage, but there have been several proposals recently, either using existing model-checking tools [10, 25] or developing new techniques [20, 21]. A recent line of research [1–5] takes the latter approach and proposes a method inspired by abstract interpretation techniques. Roughly speaking, a GTS \mathcal{R} , whose state space might be infinite, is approximated by a finite structure $\mathcal{C}(\mathcal{R})$, called *covering* of \mathcal{R} . The covering is a Petri net-like structure, called Petri graph, and it approximates \mathcal{R} in the sense that any graph G reachable in \mathcal{R} has an homomorphic image reachable in $\mathcal{C}(\mathcal{R})$. In a sense, this reduces the verification of GTSs to the verification of Petri nets. One central feature of this approach is the fact that it is a partial order reduction technique using unfoldings. That is, the interleaving of concurrent events—leading to state explosion—is avoided if possible.

* Research partially supported by the EU IST-2004-16004 SENSORIA, the MIUR PRIN 2005015824 ART, the DFG project SANDS and CRUI/DAAD VIGONI “Models based on Graph Transformation Systems: Analysis and Verification”.

In [5] a logic for the approximation approach is introduced, which is basically a propositional μ -calculus whose atomic predicates are closed formulae in a monadic second-order logic for graphs. Also, fragments of the logic are identified which are reflected by the approximations, i.e., classes of formulae which, when satisfied by the approximation, are satisfied by the original system as well. For the verification of such formulae, the logic is encoded into a μ -calculus whose atomic predicates are formulae over markings of a Petri net, allowing the reuse of existing model checking techniques for Petri nets [12].

There are other related papers working with graph logics, for instance [14]. However, most of them are based, like [5], on *propositional* temporal logics, that is, logics that do not allow to interleave temporal modalities with graph-related ones. Thus, properties like *a certain edge is never removed* are neither expressible nor verifiable. The only exceptions we are aware of are [20, 22].

In this paper we extend the approach of [5] by considering a more expressive logic that allows to interleave temporal and graphical aspects. As we shall see, our temporal graph logic combines a monadic-second order logic of graphs with the μ -calculus. Formulae of our logic are interpreted over *graph transition systems* (GTrS), inspired by algebra transition systems [15] and the formalism of [20], which are traditional transition systems together with a function mapping states into graphs and transitions into partial graph morphisms. Graph transition systems are suitable formalisms for modeling the state space of graph transformation systems and Petri graphs. We introduce a notion of approximation for GTrSs, identifying fragments of the logic whose formulae are preserved or reflected by approximations. Then we show that the GTrS of the covering, as defined in [1], is an approximation of the GTrS of the original graph transformation system, thus providing a concrete way of constructing approximations. Finally, we propose an encoding for a fragment of our logic into a Petri net logic. Our encoding is correct and complete, i.e., a Petri graph satisfies a formula exactly when the encoding of the formula is satisfied by the underlying Petri net.

Putting all this together, given a graph transformation system \mathcal{G} and a formula F in a suitable fragment of our logic, we can construct a Petri graph P which approximates \mathcal{G} , using the algorithm in [1]. Then F can be translated into a Petri net formula $[F]$, such that if N_P is the Petri net underlying P , then $N_P \models [F]$ implies $\mathcal{G} \models F$, i.e., we reduce verification over graph transformation systems to verification over Petri nets.

Section 2 introduces graphs, graph transformation systems and graph transition systems. Section 3 defines syntax and semantics of our temporal graph logic. Section 4 defines Petri graphs, the structures used for approximating graph transformation systems. Section 5 identifies fragments of the logic that are preserved or reflected by approximations. Section 6 proposes an encoding of a fragment of the logic into a Petri net logic. A last section concludes the paper and proposes further work.

2 Graph Transition Systems

An (edge-labeled) *graph* G is a tuple $G = \langle V_G, E_G, s_G, t_G, lab_G \rangle$ where V_G is a set of *nodes*, E_G is a set of *edges*, $s_G, t_G : E_G \rightarrow V_G$ are the *source* and *target* functions, and $lab_G : E_G \rightarrow \Lambda$ is a *labeling function*, where Λ is a fixed set of labels. Nodes and edges are sometimes called *items* and we shall write $X_G = E_G \cup V_G$ for the set of items of G .

The transformation of a graph into another by adding, removing or merging of items is suitably modeled by (partial) graph morphisms.

Definition 1 ((partial) graph morphism). A graph morphism $\psi : G_1 \rightarrow G_2$ is a pair of mappings $\psi_V : V_{G_1} \rightarrow V_{G_2}$, $\psi_E : E_{G_1} \rightarrow E_{G_2}$ such that $\psi_V \circ s_{G_1} = s_{G_2} \circ \psi_E$, $\psi_V \circ t_{G_1} = t_{G_2} \circ \psi_E$ and $lab_{G_1} = lab_{G_2} \circ \psi_E$. A graph morphism $\psi : G_1 \rightarrow G_2$ is injective if so are ψ_V and ψ_E ; it is edge-injective if ψ_E is injective. Edge-bijective morphisms are defined analogously. A graph G' is a subgraph of graph G , written $G' \hookrightarrow G$, if $V_{G'} \subseteq V_G$ and $E_{G'} \subseteq E_G$, and the inclusion is a graph morphism.

A partial graph morphism $\psi : G_1 \rightarrow G_2$ is a pair $\langle G'_1, \psi' \rangle$ where $G'_1 \hookrightarrow G_1$ is a subgraph of G_1 , called the domain of ψ , and $\psi' : G'_1 \rightarrow G_2$ is a graph morphism.

Graph transformation is presented in set-theoretical terms, but could be equivalently presented by using the double-pushout [7] or single-pushout [11] approaches. With respect to more general definitions, our rules can neither delete nor merge nodes, and they have a discrete interface, i.e., the interface graph contains only nodes and thus edges are never preserved. Similar restrictions are assumed in [1]. While the condition of having a discrete interface can be relaxed, the deletion and merging of nodes is quite problematic in an unfolding-based approach.

Also observe that, as it commonly happens in the algebraic approaches to graph rewriting, we consider basic graph grammars, without any distinction between terminal and non-terminal symbols and without any high-level control mechanism. We remark that, even in this basic formulation, graph grammars are Turing complete (since they can simulate string rewriting).

Definition 2 (graph transformation system). A graph transformation system (GTS) \mathcal{R} is a pair $\langle G_0, R \rangle$, where G_0 is a start graph and R is a set of rewriting rules of the form $r = \langle G_L, G_R, \alpha \rangle$, where G_L and G_R are left- and right-hand side graphs, respectively, and $\alpha : V_L \rightarrow V_R$ is an injective function.

A match of a rule r in a graph G is a morphism $\psi : G_L \rightarrow G$ that is edge-injective. The application of a rule r to a match ψ in G , resulting in a new graph H , is called a direct derivation and is written $G \xrightarrow{r, \psi} H$, where H is defined as follows. The set V_H is $V_G \uplus (V_R \setminus \alpha(V_L))$ and E_H is $(E_G \setminus \psi(E_L)) \uplus E_R$, where \uplus denotes disjoint union. The source, target and labeling functions are defined by

$$\begin{aligned} s_H(e) &= s_G(e) & t_H(e) &= t_G(e) & lab_H(e) &= lab_G(e) & \text{if } e \in (E_G \setminus \psi(E_L)), \\ s_H(e) &= \psi(s_R(e)) & t_H(e) &= \psi(t_R(e)) & lab_H(e) &= lab_R(e) & \text{if } e \in E_R, \end{aligned}$$

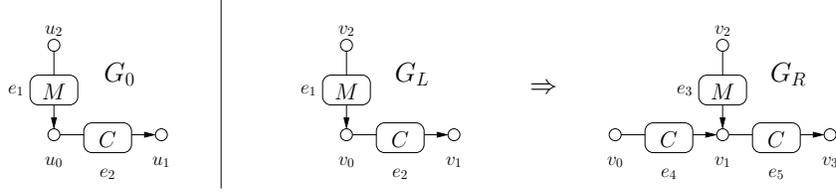


Fig. 1. A graph transformation system.

where $\bar{\psi} : V_R \rightarrow V_H$ satisfies $\bar{\psi}(\alpha(v)) = \psi(v)$ if $v \in V_L$ and $\bar{\psi}(v) = v$, otherwise.

Intuitively, the application of r to G at the match ψ first removes from G the image of the edges of L . Then the graph G is extended by adding the new nodes in G_R and the edges of G_R . All nodes in L are preserved.

A direct derivation $G \xrightarrow{r, \psi} H$ induces an obvious partial graph morphism $\tau_{G \xrightarrow{r, \psi} H} : G \rightarrow H$, injective and total on nodes, which maps items which are not deleted in G to corresponding items in H .

A *derivation* is a sequence of direct derivations starting from the start graph G_0 . We write $G_0 \xrightarrow{*} H$ if there is a derivation ending in graph H , and we denote by $\mathbf{G}_{\mathcal{R}}$ the set of all graphs reachable in \mathcal{R} , i.e., $\mathbf{G}_{\mathcal{R}} = \{G \mid G_0 \xrightarrow{*} G\}$.

Example 1. Figure 1 depicts a GTS $\mathcal{G} = \langle G_0, \{r = \langle G_L, G_R, \alpha \rangle\} \rangle$ describing a simple message passing system. The start graph G_0 consists of three nodes u_0, u_1, u_2 , one M -labeled edge e_1 , representing a *message*, and one C -labeled edge e_2 , representing a *connection*. The only rule consists of graphs G_L and G_R , and function α , which is the identity on V_L . The rule consumes the message and the connection, shifts the message to the successor node and recreates the connection. Furthermore a new C -labeled edge e_5 is created, along which the message can be passed in the next step. Note also that the source node of the message, representing its “identity”, is preserved by the rule. In the rest of the paper we shall use this as a running example.

The state space of a GTS can be represented in a natural way as a transition system where the states are the reachable graphs and a transition between two states G and H exists whenever there is a direct derivation $G \xrightarrow{r, \psi} H$, as in [3]. However, such a structure would not be sufficient to interpret the logic introduced in the next section. Informally, since temporal modalities can be interleaved with quantification (over edges), the logic allows to trace the evolution of graph items over time and thus we need to represent explicitly which items of a graph are preserved by a rewriting step. To this aim, after recalling the standard definition of transition systems, we introduce below an enriched variant called *graph transition systems*.

Definition 3 (transition system). A transition system is a tuple $M = \langle S_M, T_M, in_M, out_M, s_0^M \rangle$ where S_M is a set of states, T_M is a set of transitions,

$in_M, out_M : T_M \rightarrow S_M$ are functions mapping each transition to its start and end state, and $s_0^M \in S_M$ is the initial state. We shall sometimes write $s \xrightarrow{t} s'$ if $in_M(t) = s$ and $out_M(t) = s'$, and $s \xrightarrow{*} s'$ if there exists a (possibly empty) sequence of transitions from s to s' .

Correspondingly, a transition system morphism $h : M \rightarrow M'$ is a pair of functions $\langle h^S : S_M \rightarrow S_{M'}, h^T : T_M \rightarrow T_{M'} \rangle$ such that the initial state as well as the start and end states of all transitions are preserved, i.e., $h^S(s_0^M) = s_0^{M'}$, $h^S \circ in_M = in_{M'} \circ h^T$, and $h^S \circ out_M = out_{M'} \circ h^T$.

A *graph transition system* is defined as a transition system together with a mapping which associates a graph with each state, and an injective partial graph morphism with each transition. We use the same name that is used in [20] for different, but closely related structures. The main difference is that in our case there is a clear distinction between the states and the graphs associated to the states: This leads below to a natural notion of morphism between graph transition systems, which will play a basic role in our definition of abstraction.

Definition 4 (graph transition system). A graph transition system (*GTrS*) \mathcal{M} is a pair $\langle M, g \rangle$, where M is a transition system and g is a pair $g = \langle g^S, g^T \rangle$, where $g^S(s)$ is a graph for each state $s \in S_M$, and $g^T(t) : g^S(in_M(t)) \rightarrow g^S(out_M(t))$ is an injective partial graph morphism for each transition $t \in T_M$.

Note that the result of the application of a rule to a given match in a graph is determined only up to isomorphism, because of the use of disjoint union in the definition. Therefore, formally speaking, the state space of a GTS contains for each reachable graph G all graphs isomorphic to G as well. The next definition shows how to represent the state space of a GTS with a graph transition system (*GTrS*), where we get rid of such useless redundancy. Note that since the resulting *GTrS* is usually infinite-state, this construction is non-constructive and useless for practical purposes. We need the *GTrS* in order to define the semantics of the logic, but verification itself is done using a different method.

Definition 5 (graph transition system of a graph transformation system). Given a *GTS* $\mathcal{R} = \langle G_0, R \rangle$, a *GTrS* representing its state space, denoted by $\text{GTrS}(\mathcal{R})$, can be obtained as follows.

1. Consider first the graph transition system $\langle M, g \rangle$, where: $S_M = \mathbf{G}_{\mathcal{R}}$ (set of all graphs generated by \mathcal{R}); $s_0^M = G_0$; $T_M = \{G \xrightarrow{r, \psi} H \mid G \xrightarrow{r, \psi} H \text{ is a direct derivation of } \mathcal{R}\}$; and the mapping $g = \langle g^S, g^T \rangle$ is defined as follows: $g^S(G) = G$ and $g^T(G \xrightarrow{r, \psi} H) = \tau_{G \xrightarrow{r, \psi} H} : G \rightarrow H$.
2. Next, for each state G in S_M and for each pair $\langle r, \psi \rangle$ where r is a rule applicable to match ψ in G , choose one among the transitions leaving from G and using r and ψ , and delete from T_M all the remaining ones.
3. Finally, $\text{GTrS}(\mathcal{R})$ is defined as the graph transition sub-system reachable from the start graph.

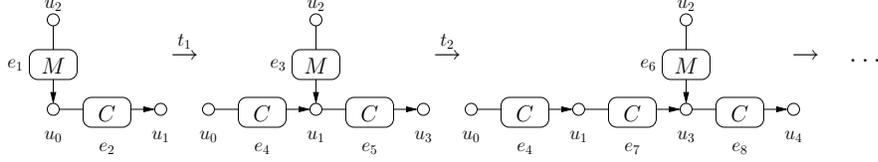


Fig. 2. A graph transition system.

Example 2. Figure 2 depicts a GTrS of the GTS depicted in Figure 1. Since state identities coincide with their corresponding graphs, the figure is simplified and we directly depict the graphs and partial graph morphisms. The leftmost state is G_0 , the initial state of both the GTS and its GTrS. Observe that for the second transition t_2 , $g^T(t_2)_V$ (the component on nodes of $g^T(t_2)$) is an inclusion, while $g^T(t_2)_E$ is partial and is only defined on the edge e_4 . All transitions correspond to different instances of the same rule.

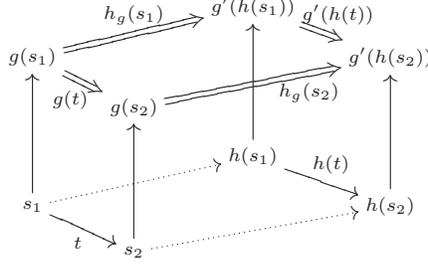
The construction described in Definition 5 is clearly non-deterministic, because of step 2. Among the possible GTrSs associated with the GTS of Figure 1, the one drawn above enjoys some desirable properties: the partial injective morphisms associated with transitions are *partial inclusions* (i.e., every item preserves its name along rewriting), and edge and node names are not reused again in the computation after they have been deleted. The interpretation of the logic of Section 3 will be defined only over GTrSs satisfying such properties, and called *unraveled GTrSs*. For any GTrS \mathcal{M} not satisfying these properties we shall consider an unraveled one which is behaviorally equivalent to \mathcal{M} , called its *unraveling*.

In order to characterize the unraveling of a GTrS we first need to introduce GTrS morphisms, which will also be used later for relating a system and its approximation. A morphism between two GTrSs consists of a morphism between the underlying transition systems, and, in addition, for each pair of related states, of a graph morphism between the graphs associated with such states. Furthermore, these graph morphisms must be consistent with the partial graph morphisms associated to the transitions.

Definition 6 (graph transition system morphism). A graph transition system morphism $h : \mathcal{M} \rightarrow \mathcal{M}'$ from $\mathcal{M} = \langle M, g \rangle$ to $\mathcal{M}' = \langle M', g' \rangle$ is a pair $\langle h_M, h_g \rangle$, where $h_M : M \rightarrow M'$ is a transition system morphism, and for each state $s \in S_M$, $h_g(s)$ is a graph morphism from $g^S(s)$ to $g'^S(h_M^S(s))$, such that the following condition is satisfied: for each transition $s_1 \xrightarrow{t} s_2 \in T_M$, $g'^T(h_M^T(t)) \circ h_g(s_1) = h_g(s_2) \circ g^T(t)$.

The diagram below illustrates the situation. The bottom square represents transition $s_1 \xrightarrow{t} s_2$ in M and its image through h_M in M' (sub- and superscripts are avoided for the sake of readability). The vertical arrows of the left

front square show how transition t is associated to a graph morphism via the g component of \mathcal{M} , and similarly for the back right square. Finally, the back left and front right sides of the top square are the components of the GTrS morphism associated to states s_1 and s_2 , and the top square is required to commute.



Definition 7 (unraveled graph transition system). A GTrS $\mathcal{M} = \langle M, g \rangle$ is unraveled whenever M is a tree, for each $t \in T_M$ the morphism $g^T(t) : g^S(\text{in}_M(t)) \rightarrow g^S(\text{out}_M(t))$ is a partial inclusion, and item names are not reused, i.e., for all $s', s'' \in S_M$, if $x \in X_{g^S(s')} \cap X_{g^S(s'')}$ there exists $s \in S_M$ such that

$x \in X_{g^S(s)} \wedge s \xrightarrow{*} s' \wedge s \xrightarrow{*} s'' \wedge g^T(s \xrightarrow{*} s')(x) = x \wedge g^T(s \xrightarrow{*} s'')(x) = x$, where $g^T(s \xrightarrow{*} s')$ is the composition of the partial morphisms associated with the transitions in $s \xrightarrow{*} s'$, which is uniquely determined since M is a tree.

An unraveling of a GTrS $\mathcal{M} = \langle M, g \rangle$ is a pair $\langle \mathcal{M}', h \rangle$ where \mathcal{M}' is an unraveled GTrS and $h = \langle h_M, h_g \rangle : \mathcal{M}' \rightarrow \mathcal{M}$ is a GTrS morphism, satisfying:

1. for each $s \in S_{\mathcal{M}'}$, $h_g(s) : g'^S(s) \rightarrow g^S(h_M^S(s))$ is an isomorphism;
2. for each $s \in S_{\mathcal{M}'}$ and transition $h_M^S(s) \xrightarrow{t'} s''$ in \mathcal{M}' , there is a transition $s \xrightarrow{t} s'$ in \mathcal{M} such that $h_M^T(t) = t'$ (and thus $h_M^S(s') = s''$).

Proposition 1 (unraveling of a GTrS). Any GTrS admits an unraveling.

The conditions defining an unraveled GTrS \mathcal{M} ensure that taking the union of all the graphs associated to the states in S_M , we obtain a well-defined graph. In fact, given any two states s and s' and an edge $e \in E_{g^S(s)} \cap E_{g^S(s')}$, the source, target and label of e coincide in $g^S(s)$ and $g^S(s')$. We shall denote the components of this “universe” graph as $\langle V_{\mathcal{M}}, E_{\mathcal{M}}, s_{\mathcal{M}}, t_{\mathcal{M}}, \text{lab}_{\mathcal{M}} \rangle$, where $V_{\mathcal{M}} = \bigcup_{s \in S_M} V_{g^S(s)}$, $E_{\mathcal{M}} = \bigcup_{s \in S_M} E_{g^S(s)}$, $s_{\mathcal{M}}(e) = s_{g^S(s)}(e)$ if $e \in g^S(s)$, and similarly for $t_{\mathcal{M}}$ and $\text{lab}_{\mathcal{M}}$.

3 A Temporal Graph Logic for Graph Transformation Systems

We now define syntax and semantics of our temporal graph logic, that extends the logic $\mu\mathcal{L}2$ of [3]. The logic is based both on the μ -calculus [6] and on second-order graph logic [8]. Let V_x, V_X, V_Z be sets of first- and second-order edge variables and propositional variables, respectively.

Definition 8 (syntax). The logic $\mu\mathcal{G}2$ is given by the set of all formulae generated by:

$$F ::= \eta(x) = \eta'(y) \mid x = y \mid l(x) = a \mid \neg F \mid F \vee F \mid \exists x.F \mid \exists X.F \mid \\ x \in X \mid Z \mid \diamond F \mid \mu Z.F$$

where $\eta, \eta' \in \{s, t\}$ (standing for source and target), $x, y \in V_x$, $X \in V_X$, $a \in \Lambda$ and $Z \in V_Z$. Furthermore $\diamond F$ is the (existential) next-step operator. The letter μ denotes the least fixed-point operator. As usual the formula $\mu Z.F$ can be formed only if all occurrences of Z in F are positive, i.e., they fall under an even number of negations. In the following we will use some (redundant) connectives like \wedge, \forall, \square and ν (greatest fixed-point), defined as usual. We denote by $\mu\mathcal{G}1$ its first-order fragment, where second-order edge variables and quantification are not allowed.

Definition 9 (semantics of $\mu\mathcal{G}2$). Let $\mathcal{M} = \langle M, g \rangle$ be an unraveled GTrS. The semantics of temporal graph formulae is given by an evaluation function mapping closed formulae into subsets of $S_{\mathcal{M}}$, i.e., the states that satisfy the formula. We shall define a mapping $\llbracket \cdot \rrbracket_{\sigma}^{\mathcal{M}} : \mu\mathcal{G}2 \rightarrow 2^{S_{\mathcal{M}}}$, where σ is an environment, i.e., a tuple $\sigma = \langle \sigma_x, \sigma_X, \sigma_Z \rangle$ of mappings from first- and second-order edge variables into edges and edge sets, respectively, and from propositional variables into subsets of $S_{\mathcal{M}}$. More precisely, $\sigma_x : V_x \rightarrow E_{\mathcal{M}}$, $\sigma_X : V_X \rightarrow 2^{E_{\mathcal{M}}}$ and $\sigma_Z : V_Z \rightarrow 2^{S_{\mathcal{M}}}$, where $E_{\mathcal{M}}$ is the set of all edge names used in \mathcal{M} . When \mathcal{M} is implicit, we simply write $\llbracket \cdot \rrbracket_{\sigma}$.

$$\begin{aligned} \llbracket \eta(x) = \eta'(y) \rrbracket_{\sigma} &= \llbracket \eta_{\mathcal{M}}(\sigma_x(x)) = \eta'_{\mathcal{M}}(\sigma_x(y)) \rrbracket & \llbracket x = y \rrbracket_{\sigma} &= \llbracket \sigma_x(x) = \sigma_x(y) \rrbracket \\ \llbracket l(x) = a \rrbracket_{\sigma} &= \llbracket \text{lab}_{\mathcal{M}}(\sigma_x(x)) = a \rrbracket & \llbracket y \in Y \rrbracket_{\sigma} &= \llbracket \sigma_x(y) \in \sigma_X(Y) \rrbracket \\ \llbracket \neg F \rrbracket_{\sigma} &= S_{\mathcal{M}} \setminus \llbracket F \rrbracket_{\sigma} & \llbracket F_1 \vee F_2 \rrbracket_{\sigma} &= \llbracket F_1 \rrbracket_{\sigma} \cup \llbracket F_2 \rrbracket_{\sigma} \\ \llbracket Z \rrbracket_{\sigma} &= \sigma_Z(Z) & \llbracket \mu Z.F \rrbracket_{\sigma} &= \mathbf{lfp}(\lambda v. \llbracket F \rrbracket_{\sigma[v/Z]}) \\ \llbracket \diamond F \rrbracket_{\sigma} &= \{s \in S_{\mathcal{M}} \mid \exists s', t. s \xrightarrow{t} s' \wedge s' \in \llbracket F \rrbracket_{\sigma}\} \\ \llbracket \exists x.F \rrbracket_{\sigma} &= \{s \in S_{\mathcal{M}} \mid \exists e \in E_{g(s)}. s \in \llbracket F \rrbracket_{\sigma[e/x]}\} \\ \llbracket \exists X.F \rrbracket_{\sigma} &= \{s \in S_{\mathcal{M}} \mid \exists E \subseteq E_{g(s)}. s \in \llbracket F \rrbracket_{\sigma[E/X]}\} \end{aligned}$$

where $\llbracket \cdot \rrbracket$ maps true and false to $S_{\mathcal{M}}$ and \emptyset , respectively, $v \in 2^{S_{\mathcal{M}}}$, and $\mathbf{lfp}(f)$ denotes the least fixed-point of the function f .

In particular, if F is a closed formula, we say that \mathcal{M} satisfies F and write $\mathcal{M} \models F$, if $s_0 \in \llbracket F \rrbracket_{\sigma}$, where σ is the empty environment. Finally, we say that a GTS $\mathcal{R} = \langle G_0, R \rangle$ satisfies a closed formula F , written $\mathcal{R} \models F$, if the unraveling of $GTrS(\mathcal{R})$ satisfies F .

The restriction to formulae where all occurrences of propositional variables are positive guarantees every possible function $\lambda v. \llbracket F \rrbracket_{\sigma[v/Z]}$ to be monotonic. Thus, by Knaster-Tarski theorem, fixed-points are well-defined.

Note that using unravelled GTrS is crucial for the definition of the semantics of the logic: items can be easily tracked since their identity is preserved and names are never reused. This allows to remember also the identities of deleted items, differently from what happens in the semantics given in [20, 22].

Example 3. The following formula states that no M -labeled message edge is preserved by any transition: **M-consumed** $\equiv \neg\exists x.(l(x) = M \wedge \diamond\exists y.x = y)$. The fact that this property holds in any reachable state is expressed by the formula: **always-M-consumed** $\equiv \nu Z.(\mathbf{M-consumed} \wedge \square Z)$. It is easy to see that **M-consumed** is satisfied by any state of the unraveled GTrS in Fig. 2, and thus $\mathcal{G} \models \mathbf{always-M-consumed}$, where \mathcal{G} is the GTS of Fig. 1.

The formula **M-moves** $\equiv \neg\exists x.(l(x) = M \wedge \diamond(\exists y.(l(y) = M \wedge t(y) = t(x) \wedge s(y) = s(x))))$ states that messages always move, i.e., there is no message edge such that in the next state there is another message edge with the same identity (i.e., source nodes coincide) attached to the same target node. And we can require this property to hold in every reachable state: **always-M-moves** $\equiv \nu Z.(\mathbf{M-moves} \wedge \square Z)$. Again, the GTS \mathcal{G} satisfies this property. A GTS in which the message would at some point cross a “looping connection” or with more than one message would violate the formula.

4 Approximating GTSs with Petri graphs

In the verification approach proposed in [1, 3–5] *Petri graphs*, structures consisting of a Petri net and a graph component, have been introduced. They are used to represent finite approximations of the (usually infinite) unfolding of a GTS, on which to verify certain properties of the original system. Furthermore they provide a bridge to Petri net theory, allowing to reuse verification techniques developed for nets: a property expressed as a formula in the graph logic can be translated into an equivalent multiset formula to be verified on the net underlying the Petri graph. Here we shall concentrate on this latter aspect. We will not treat instead the construction of finite Petri graphs over-approximating GTSs, presented in [1, 4] also for varying degrees of precision, recently enriched with a technique for counterexample-guided abstraction refinement [18], and for which the verification tool AUGUR (http://www.ti.inf.uni-due.de/research/augur_1/) has been developed.

Before introducing Petri graphs we need some definitions. Given a set A we will denote by A^\oplus the free commutative monoid over A , whose elements will be called *multisets* over A . In the sequel we will sometimes identify A^\oplus with the set of functions $m : A \rightarrow \mathbb{N}$ such that the set $\{a \in A \mid m(a) \neq 0\}$ is finite. E.g., in particular, $m(a)$ denotes the multiplicity of an element a in the multiset m . Sometimes a multiset will be identified with the underlying set, writing, e.g., $a \in m$ for $m(a) \neq 0$. Given a function $f : A \rightarrow B$, by $f^\oplus : A^\oplus \rightarrow B^\oplus$ we denote its monoidal extension, i.e., $f^\oplus(m)(b) = \sum_{f(a)=b} m(a)$ for every $b \in B$.

Definition 10 (Petri nets and Petri graphs). A (Place/Transition) Petri net is a tuple $N = \langle S_N, T_N, \bullet(\cdot), (\cdot)^\bullet, m_0 \rangle$, where S_N is a set of places, T_N is a set of transitions, $\bullet(\cdot), (\cdot)^\bullet : T_N \rightarrow S_N^\oplus$ determine for each transition its pre-set and post-set, and $m_0 \in S_N^\oplus$ is the initial marking. A transition t is enabled at a

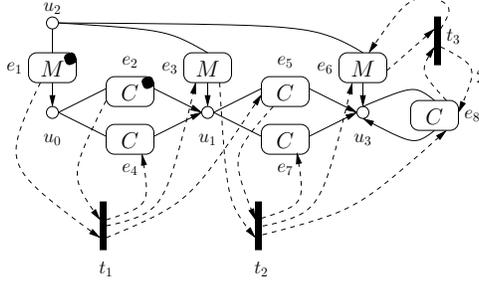


Fig. 3. A Petri graph.

marking $m \in S_N^\oplus$ if $\bullet t \leq m$; in this case the transition can be fired at m , written $m[t]m'$, and the resulting marking is $m' = m - \bullet t + t \bullet$.¹

A Petri graph P over a GTS $\mathcal{R} = \langle G_0, R \rangle$ is a tuple $\langle G, N, p \rangle$ where (1) G is a graph (sometimes called the template graph); (2) N is a Petri net such that (2.1) $S_N = E_G$, i.e., the set of places is the set of edges of G ; (2.2) there is a graph morphism $\psi : G_0 \rightarrow G$; (2.3) the initial marking $m_0 \in E_G^\oplus$ properly corresponds to the start graph of \mathcal{R} , i.e., $m_0 = \psi^\oplus(E_{G_0})$; (3) $p : T_N \rightarrow R$ is a labeling function mapping each transition to a rule of \mathcal{R} , such that (3.1) for each transition $t \in T_N$, its pre- and post-sets $\bullet t$ and $t \bullet$ properly correspond to the left- and right-hand side graphs of $p(t)$. A marking m is said to be reachable in P if there is a (possibly empty) sequence of firing $m_0[t_1]m_1[t_2] \dots [t_n]m$ in the underlying net N . The set of reachable markings of P is denoted by \mathbf{M}_P .

Example 4. Figure 3 depicts a Petri graph over the GTS \mathcal{G} of our running example. It has been computed by the tool AUGUR as the covering of depth 1. As edges are places, the boxes representing edges can include tokens, which here represent the initial marking. Transitions are depicted as black rectangles. The pre-sets (resp. post-sets) of transitions are represented by dotted edges from edge places to transitions or vice versa. Note that all three transitions are instances of rule r , the only rule of \mathcal{G} , and that there is indeed a morphism from G_0 to the template graph, such that the image of this morphism (e_1, e_2) corresponds to the depicted (initial) marking.

A marking m of a Petri graph can be seen as an abstract representation of a graph. The intuition is that every token of an edge represents an instance of the corresponding template edge.

Definition 11 (graph generated by a marking). Let $P = \langle G, N, p \rangle$ be a Petri graph and let $m \in E_G^\oplus$ be a marking of N . The graph generated by m , denoted by $\text{graph}_P(m)$, is the graph H defined as follows: $V_H = V_G$, $E_H = \{\langle e, i \rangle \mid e \in m \wedge 1 \leq i \leq m(e)\}$, $s_H(\langle e, i \rangle) = s_G(e)$, $t_H(\langle e, i \rangle) = t_G(e)$ and

¹ Operations on markings are computed pointwise on the coefficients: e.g., $m_1 \leq m_2$ iff $m_1(s) \leq m_2(s)$ for all $s \in S_N$, and $(m_1 + m_2)(s) = m_1(s) + m_2(s)$.

$lab_H(\langle e, i \rangle) = lab_G(e)$. In the following ψ_m denotes the obvious graph morphism from $graph_P(m)$ into G , which is the identity on nodes and which satisfies $\psi_{mE}(\langle e, i \rangle) = e \in E_G$.

Example 5. The marking of the Petri graph depicted in Figure 3 generates the leftmost graph of Figure 4. The remaining graphs are generated by other reachable markings.

To each Petri graph P we can associate a GTrS, as shown in the next definition. As the reader would expect, each marking m is a state and the associated graph is $graph_P(m)$. However, each transition of P corresponds in general to a set of transitions in the GTrS, representing the different ways of preserving the edges. To see why this is necessary consider the Petri graph of our running example. Transition t_3 consumes a token from edge place e_8 . Now assume that the marking m is such that $m(e_8) = 2$, i.e., there are two tokens in e_8 . Such a marking is indeed reachable. In this case one has to consider two transitions in the transition system: one consuming edge $\langle e_8, 1 \rangle$ and the other consuming $\langle e_8, 2 \rangle$. The intuitive idea of having different ways of preserving edges is formalized by the notion of *significant preservations*.

Definition 12 (significant preservations). Let $P = \langle G, N, p \rangle$ be a Petri graph, m, m' be markings and t a transition such that $m[t]m'$. We denote by $SP(m, t)$ the set of significant preservations, which contains the possible different subsets of edges in $graph_P(m)$ which are not deleted by a firing of t , i.e., $SP(m, t) = \{E_{graph_P(m)} - Y : Y \subseteq E_{graph_P(m)} \wedge \psi_m^\oplus(Y) = \bullet t\}$.

Definition 13 (GTrS by a Petri graph). The GTrS generated by a Petri graph $P = \langle G, N, p \rangle$, denoted by $GTrS(P)$, is $\langle M, g \rangle$ where

- S_M is the set of markings reachable in P , i.e., $S_M = \mathbf{M}_P$;
- $T_M = \{\langle m, t, X, m' \rangle \mid m[t]m' \text{ and } X \in SP(m, t)\}$;
- $in(\langle m, t, X, m' \rangle) = m$ and $out(\langle m, t, X, m' \rangle) = m'$ for $\langle m, t, X, m' \rangle \in T_M$;
- $s_0^M = m_0$;
- $g^S(m) = graph_P(m)$;
- $g^T(\langle m, t, X, m' \rangle) = f_{m,t,X}$, where $f_{m,t,X} : graph_P(m) \rightarrow graph_P(m')$ is any injective partial graph morphism which is the identity over nodes, and whose domain over edges is exactly X ; for example, a concrete definition over edges can be $f_{m,t,X}^E(\langle e, i \rangle) = \langle e, k \rangle$ if $k = |\langle e, j \rangle \in X : j \leq i|$.

Example 6. Figure 4 illustrates the GTrS associated to the Petri graph of our running example. For the sake of simplicity only the underlying graphs and partial graph morphisms are depicted. The leftmost state corresponds to the initial marking $m_0 = \{e_1, e_2\}$, while the next one corresponds to marking $m_1 = \{e_3, e_4, e_5\}$ reachable after firing t_1 . In the third graph a looping C -edge is introduced, due to over-approximation. The most interesting transitions are t'_4, t''_4 , both leaving marking m_3 . Both have the form $\langle m_3, t_3, X, m_4 \rangle$, where X can either be $\{\langle e_4, 1 \rangle, \langle e_7, 1 \rangle, \langle e_8, 1 \rangle\}$ or $\{\langle e_4, 1 \rangle, \langle e_7, 1 \rangle, \langle e_8, 2 \rangle\}$. These transitions represent different ways of consuming an instance of edge place e_8 .

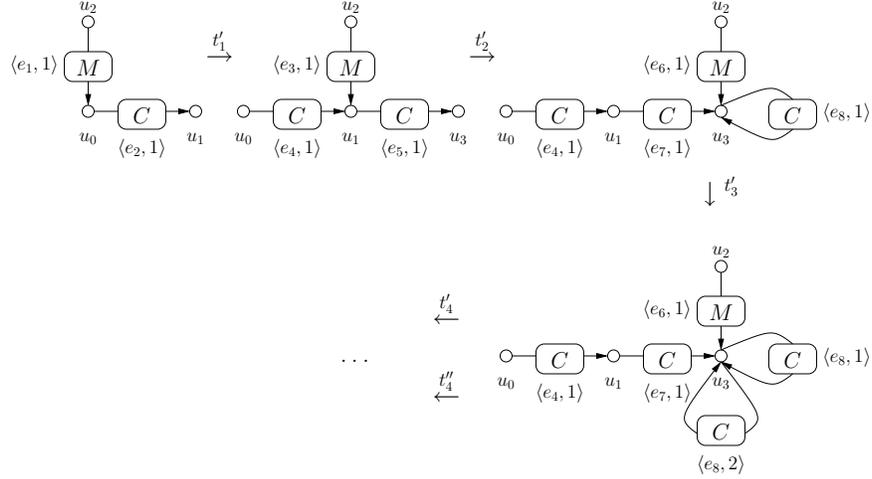


Fig. 4. A graph transition system of a Petri graph.

Definition 14 (approximation). Let $\mathcal{R} = \langle G_0, R \rangle$ be a GTS, let $P = \langle G, N, p \rangle$ be a Petri graph and let $\mathcal{M}_{\mathcal{R}}$ and \mathcal{M}_P be the unravelings of the GTrSs generated by \mathcal{R} and P , respectively. We say that P is an approximation of \mathcal{R} if there is a GTrS morphism $\langle h_M, h_g \rangle : \mathcal{M}_{\mathcal{R}} \rightarrow \mathcal{M}_P$ such that for each state $s \in S_{\mathcal{M}_{\mathcal{R}}}$, the graph morphism $h_g(s)$ is edge-bijective.

It is easy to see that this notion of approximation implies a simulation in Milner's sense: if the original system can make a transition t from a graph G to a graph H , then the approximation can simulate it with a transition from a graph G' to a graph H' via $h(t)$. Additionally, we require that there must be edge-bijective morphisms from G to G' and from H to H' , a property which will be crucial for determining fragments of our logic that are preserved or reflected by the approximation.

Example 7. Consider our running example. It is easy to see that the Petri graph in Fig. 3 approximates the GTS in Fig. 1: there exist several morphisms with edge-bijective components from the GTrS of Fig. 2 to the unraveling of the GTrS of Fig. 4.

As already mentioned, given a GTS $\mathcal{R} = \langle G_0, R \rangle$, an algorithm proposed in [1, 4] constructs a Petri graph associated to \mathcal{R} , called its *covering* and denoted by $\mathcal{C}(\mathcal{R})$. The covering is an approximation of \mathcal{R} in the sense stated above.

Proposition 2 (covering approximates). Let $\mathcal{R} = \langle G_0, R \rangle$ be a graph transformation system. Then the covering $\mathcal{C}(\mathcal{R}) = \langle G, N, p \rangle$ of \mathcal{R} is an approximation of \mathcal{R} .

5 Preservation and Reflection of Formulae

In this section we introduce a type system over graph formulae in $\mu\mathcal{G}2$, generalizing the one in [5], which characterizes subclasses of formulae preserved or reflected by approximations. More precisely, the type system may assign to a formula F the type “ \rightarrow ”, meaning that F is preserved by approximations, or the type “ \leftarrow ”, meaning that F is reflected by approximations. Note especially that since the approximation may merge nodes, formulae checking the identity of nodes are preserved, but not reflected.

Definition 15 (reflected/preserved formulae). *The typing rules are given by*

$$\eta(x) = \eta'(y) : \rightarrow \quad x = y, l(x) = a, x \in X, Z : \leftrightarrow$$

$$\frac{F : d}{\neg F : d^{-1}} \quad \frac{F_1, F_2 : d}{F_1 \vee F_2 : d} \quad \frac{F : d}{\exists x.F : d} \quad \frac{F : d}{\exists X.F : d} \quad \frac{F : \rightarrow}{\diamond F : \rightarrow} \quad \frac{F : \leftarrow}{\square F : \leftarrow} \quad \frac{F : d}{\mu Z.F : d}$$

where it is intended that $\rightarrow^{-1} = \leftarrow$ and $\leftarrow^{-1} = \rightarrow$. Moreover $F : \leftrightarrow$ is a shortcut for $F : \rightarrow$ and $F : \leftarrow$, while $F_1, F_2 : d$ stands for $F_1 : d$ and $F_2 : d$.

The type system can be shown to be correct in the following sense (see also [19]).

Proposition 3 (preservation and reflection). *Let \mathcal{M} and \mathcal{M}' be two unraveled GTrSs such that there is a morphism $\langle h_M, h_g \rangle : \mathcal{M} \rightarrow \mathcal{M}'$ having all h_g components edge-bijective. Then for each closed formula $F \in \mu\mathcal{G}2$ we have (i) if $F : \leftarrow$ then $\mathcal{M}' \models F$ implies $\mathcal{M} \models F$ and (ii) if $F : \rightarrow$ then $\mathcal{M} \models F$ implies $\mathcal{M}' \models F$.*

Not all formulae that are preserved respectively reflected are recognized by the above type system. A result of [5] shows that this incompleteness is a fundamental problem, due to the undecidability of reflection and preservation.

Example 8. Observe that **always-M-consumed** : \leftarrow and hence approximations reflect this property. Indeed the unraveling of the GTrS of the Petri graph of our running example (see Fig. 4) satisfies the property, and so does the original GTrS. Also, formula **always-M-moves** is classified as reflecting: however, in this case the GTrS of Fig. 2 satisfies this property, but the unraveling of the GTrS of Figure 4 does not, by the presence of a connection loop e_8 .

Since the covering provides an approximation of the original GTS, the theorem above applies. For a Petri graph P and a closed formula $F \in \mu\mathcal{G}2$, we shall write $P \models F$ if the unraveling of the graph transition system $GTrS(P)$ generated by P satisfies F .

Corollary 1 (covering preserves and approximates). *Let \mathcal{R} be a GTS and let $F \in \mu\mathcal{G}2$ be a closed formula. Then we have (i) if $F : \leftarrow$ then $\mathcal{C}(\mathcal{R}) \models F$ implies $\mathcal{R} \models F$ and (ii) if $F : \rightarrow$ then $\mathcal{R} \models F$ implies $\mathcal{C}(\mathcal{R}) \models F$.*

6 From Temporal Graph Logics to Petri Net Logics

In this section we show how the first-order fragment of $\mu\mathcal{G}2$ can be encoded into a temporal logic for Petri nets, in a way that the Petri net underlying a Petri graph satisfies the encoding of a formula exactly when the Petri graph satisfies the original $\mu\mathcal{G}2$ formula.

Let V_Z be a set of propositional variables and N_P, N_T sets from which place and transition names are drawn, respectively.

Definition 16 (Petri net logic syntax). *The syntax of the Petri net logic \mathcal{P} is given by the following grammar, where $p \in N_P, t \in N_T, c \in \mathbb{N}$ and $Z \in V_Z$:*

$$\phi ::= \#p \leq c \mid \phi \vee \phi \mid \neg\phi \mid Z \mid \mu Z.\phi \mid \langle t \rangle \phi.$$

The semantics is mostly standard and given by a mapping $\llbracket \cdot \rrbracket_\sigma^P : \mathcal{P} \rightarrow 2^{\mathbf{M}_P}$ mapping formulae into sets of markings, where $\sigma : \text{free}_Z \rightarrow 2^{\mathbf{M}_P}$ is an environment mapping propositional variables into sets of markings. We sometimes use a satisfaction relation $\models_\sigma \subseteq \mathbf{M}_P \times \mathcal{P}$, where $m \models_\sigma \phi$ whenever $m \in \llbracket \phi \rrbracket_\sigma^P$.

As an example, $\#e \leq c$ is satisfied by markings m , where $m(e) \leq c$, i.e., markings where the number of tokens in place e is less than or equal to c . Next-time modalities are enriched with transition labels with the following meaning: $\langle t \rangle \phi$ is satisfied by markings from which transition t can be fired leading to a marking that satisfies ϕ .

The encoding $\llbracket \cdot \rrbracket$, which maps formulae of $\mu\mathcal{G}1$ into formulae of the logic \mathcal{P} , is based on the following observation: Every graph $\text{graph}_P(m)$ for some marking m of P can be generated from the finite template graph G in the following way: some edges of G might be removed and some edges be multiplied, generating several parallel instances of the same template edge. Whenever a formula has two free variables x, y and $\text{graph}_P(m)$ has n parallel instances e_1, \dots, e_n of the same edge, it is not necessary to associate x and y to edges in all possible ways, but it is sufficient to remember whether x and y are mapped or not mapped to the same edge. Hence, in the encoding of a formula F , we keep track of the following information: a partition Q on the free variables $\text{free}(F)$, telling which variables are mapped to the same edge, and a mapping ρ from $\text{free}(F)$ to the edges of G , with $\rho(x) = e$ meaning that x will be instantiated with an instance of the template edge e . When encoding an existential quantifier $\exists x$, we form a disjunction over all the possibilities we have in choosing such an x : either x is instantiated with the same edge as another free variable y , and thus x and y are in the same class of the partition Q ; or x is mapped to a new instance of an edge in G , and thus a new set $\{x\}$ is added to Q , adding a suitable predicate which ensures that enough edges are available.

This is enough for the logic of [5], where interleaving of temporal operators and edge quantifiers is not allowed. Here we have to consider the case in which temporal modalities are nested into edge quantification. The main problem is that an edge where some variables have been mapped can be removed by a transition and thus, when encoding quantification, one must be careful in avoiding to

instantiate a variable with the class of a removed edge. This is faced by recording in a set R the classes corresponding to *removed* edges.

Before we define the encoding we need some definitions. An equivalence relation Q over a set A will be represented as a partition $Q \subseteq 2^A$, where every element represents an equivalence class. We will write xQy whenever x, y are in the same equivalence class $k \in Q$. Furthermore we assume that each equivalence Q is associated with a function $rep : Q \rightarrow A$ which assigns a representative to every equivalence class. The encoding given below is independent of any specific choice of representatives. Given a function $f : A \rightarrow B$ such that $f(a) = f(a')$ for all $a, a' \in A$ with aQa' , we shall often write $f(k)$ for $f(rep(k))$; furthermore, for any $b \in B$ we define $n_{Q,f}(b) = |\{k \in Q \mid f(k) = b\}|$, i.e., the number of classes in the partition Q that are mapped to b .

We next define the encoding, concentrating first on the fragment without fixed-point operators.

Definition 17 (encoding for the fixed-point free first-order logic). *Let $P = \langle G, N, \rho \rangle$ be a Petri graph, F be a fixed-point-free $\mu\mathcal{G}1$ formula, $\rho : free(F) \rightarrow E_G$ and $Q \subseteq 2^{free(F)}$ be an equivalence relation, $R \subseteq Q$ and xQy implies $\rho(x) = \rho(y)$ for all $x, y \in free(F)$. The encoding $[\cdot] : \mu\mathcal{G}1 \rightarrow \mathcal{P}$ is defined as follows:*

$$\begin{aligned}
[\neg F, \rho, Q, R] &= \neg[F, \rho, Q, R] \\
[F_1 \vee F_2, \rho, Q, R] &= [F_1, \rho, Q, R] \vee [F_2, \rho, Q, R] \\
[x = y, \rho, Q, R] &= \begin{cases} \text{true} & \text{if } xQy \\ \text{false} & \text{otherwise} \end{cases} \\
[l(x) = a, \rho, Q, R] &= \begin{cases} \text{true} & \text{if } lab_G(\rho(x)) = a \\ \text{false} & \text{otherwise} \end{cases} \\
[s(x) = s(y), \rho, Q, R] &= \begin{cases} \text{true} & \text{if } s_G(\rho(x)) = s_G(\rho(y)) \\ \text{false} & \text{otherwise} \end{cases} \\
&\quad \text{analogously for } t(x) = t(y) \text{ and } s(x) = t(y) \\
[\exists x.F, \rho, Q, R] &= \bigvee_{k \in Q \setminus R} [F, \rho \cup \{x \mapsto \rho(k)\}, Q \setminus \{k\} \cup \{k \cup \{x\}\}, R] \vee \\
&\quad \bigvee_{e \in E_G} ([F, \rho \cup \{x \mapsto e\}, Q \cup \{\{x\}\}, R] \\
&\quad \wedge (\#e \geq n_{Q \setminus R, \rho}(e) + 1)) \\
[\diamond F, \rho, Q, R] &= \bigvee_{t \in T_N} \bigvee_{R' \in S_{R,t}} (\bigwedge_{e \in \bullet t} (\#e \geq n_{Q \setminus (R \cup R'), \rho}(e) + \bullet t(e)) \wedge \\
&\quad \wedge \langle t \rangle [F, \rho, Q, R \cup R']) \\
[Z, \rho, Q, R] &= Z
\end{aligned}$$

where $S_{R,t}$ abbreviates $\{R' \in 2^{(Q \setminus R)} \mid (\rho \circ rep)^\oplus(R') \leq \bullet t\}$.

If F is closed, we define $[F]$ to be $[F, \emptyset, \emptyset, \emptyset]$. The main novelty with respect to [5] is the encoding of formulae $\diamond F$ involving the next-time operator. In order to see if there is a transition after which F holds we examine the possible transitions t of the Petri graph, and hence the disjunction amongst all $t \in T_N$ arises. Concerning the removed edges, after the firing of a transition t several

cases may apply since edges corresponding to places in the pre-set of t may be preserved or consumed, depending on the number of tokens in such places.

All the cases that have to be considered are defined by $S_{R,t}$. In words, $S_{R,t}$ contains sets of equivalence classes from $Q \setminus R$ that correspond to places in the pre-set of t , not exceeding the number of tokens removed by t from each place, i.e., if a transition consumes n tokens from a place e we shall not consider the case in which more than n equivalence classes mapped to e are consumed.

When considering the consumption of one of such set R' of equivalence classes, we have to ensure that equivalence classes not included in R' can actually be preserved. This can only happen if there are enough tokens ($\bigwedge_{e \in \bullet t} \#e \geq n_{Q \setminus (R \cup R'), \rho}(e) + \bullet t(e)$), i.e., if in each place e in the pre-set of t the number of tokens is greater or equal to the number of tokens removed by t from e plus the number of equivalence classes mapped to e that will be preserved.

Example 9. To clarify this point consider an example in which we are treating a transition t whose pre-set is $\{e, e', e'\}$ and we have that Q is $\{k_1, k_2, k_3\}$, R is empty and ρ maps k_1, k_2 to e and k_3 to e' . In this case we have that $S_{R,t}$ is $\{\emptyset, \{k_1\}, \{k_2\}, \{k_1, k_3\}, \{k_2, k_3\}\}$. Note that $\{k_1, k_2, k_3\}$ does not belong to $S_{R,t}$ because the in-degree of e for t is just 1. Now let us consider the requirements on the places for some of the elements of $S_{R,t}$. For instance, for $\{k_1\}$ we need $\#e \geq 2 \wedge \#e' \geq 3$ in order to be able to preserve k_2 and k_3 , while for $\{k_1, k_3\}$ we need $\#e \geq 2$ to preserve k_2 .

After fixing an R' and setting the requirement on the number of tokens in the pre-set of t we have only to state that formula F holds under the new configuration, i.e., where R' is added to the set of removed equivalence classes.

Example 10. For the Petri graph in Fig. 3 formula **[M-consumed]** has the following form:

$$\neg \bigvee_{(i,j) \in \mathcal{L}} \left((\#e_i \geq 2 \wedge \#e_j \geq 1) \vee \left(\#e_i \geq 1 \wedge \bigvee_{(k,\ell) \in \mathcal{L} - \{(i,j)\}} (\#e_k \geq 1 \wedge \#e_\ell \geq 1) \right) \right)$$

where $\mathcal{L} = \{(1, 2), (3, 5), (6, 8)\}$ (pairs of indices of edges which form the pre-set of a transition). Intuitively, the two disjuncts above encode the fact that an M -edge can be preserved if there is an enabled transition with more than one token in the M -edge in its pre-set (left disjunct), or there is a token in an M -edge and a transition which does not consume this token is enabled (right disjunct).

In the worst case there can be an exponential blowup in the size of the encoded formula. But at the same time, the resulting formula can often be greatly simplified, even on-the-fly.

The encoding for general formulae of $\mu\mathcal{G}1$, possibly including fixed-point operators requires additional effort. Suppose we want to express that there is an edge x and a computation that never consumes x , i.e., $F \equiv \exists x. \nu Z. \exists y. (x = y \wedge \diamond Z)$. Now, if we try to encode F we encounter a problem: sometimes Z

should be evaluated in a context where the equivalence class of x is preserved and sometimes in one where x is consumed.

In order to solve this, we exploit a property of fixed-points, namely that unfolding an occurrence of the variable of a fixed-point formula results in an equivalent formula. More formally, $\mu Z.F$ is equivalent to $\mu Z.F'$, where F' is the same as F except that some free occurrences of Z are substituted by $\mu Z'.F[Z'/Z]$, where Z' is a fresh variable. As we shall see, unfolded fixed-points will be evaluated in different contexts and a syntactic restriction will guarantee termination of the encoding.

This is handled in the encoding of the next-step operator where in the case of partition-consuming transitions we use the unfolding of F , denoted by $uf(F, R)$. The unfolding is formally defined as follows. Let $fp(Z)$ denote the fixed-point formula corresponding to a propositional variable Z and let $\{Z_1^F, \dots, Z_n^F\}$ denote the set of propositional variables appearing free in F . Then $uf(F, R)$ is defined as $F\{fp(Z_1^F)/Z_1^F, \dots, fp(Z_n^F)/Z_n^F\}$, i.e., each variable is substituted by the corresponding fixed-point formula, if R is not empty, otherwise $uf(F, R)$ is just F . The idea is that if no equivalence class is consumed the unfolding is not necessary. Of course, every propositional and edge variable must be renamed in the unfolding.

Formally, the encoding in presence of fixed-point operators is defined as follows.

Definition 18 (encoding for first-order logic). *Let $P = \langle G, N, p \rangle$ be a Petri graph, $F \in \mu\mathcal{G}1$, ρ , Q , R as in Definition 17. The encoding $[\cdot] : \mu\mathcal{G}1 \rightarrow \mathcal{P}$ is defined as in Definition 17, but for the clause of the next-step operator which becomes (a) below and the new clause for fixed-point operators (b):*

$$[\diamond F, \rho, Q, R] = \bigvee_{t \in T_N} \bigvee_{R' \in S_{R,t}} \left(\bigwedge_{e \in \bullet t} (\#e \geq n_{Q \setminus (R \cup R'), \rho}(e) + \bullet t(e)) \wedge \right. \quad (a)$$

$$\left. \wedge \langle t \rangle [uf(F, R'), \rho, Q, R \cup R'] \right)$$

$$[\mu Z.\phi, \rho, Q, R] = \mu Z. [\phi, \rho, Q, R] \quad (b)$$

In order to guarantee termination of the encoding, we have to forbid formulae in which propositional variables appear free under the scope of an edge quantifier. To see why this is necessary, one can apply the encoding to formula $\nu Z. \exists x. \diamond (Z \wedge \exists y. y = x)$, expressing that there is an infinite computation where in every state there is at least one edge that is preserved in the next state. With the restriction mentioned the encoding is guaranteed to terminate, since in this case the set Q will not increase, hence set $S_{R,t}$ will decrease and at some point the chosen R' must be empty. It is an open question whether this syntactic restriction involves a loss of expressive power.

Proposition 4 (finite encoding). *Let (G, N, p) be a Petri graph, $F \in \mu\mathcal{G}1$ such that no propositional variable appears free under the scope of an edge quantifier. Then $[F, \emptyset, \emptyset, \emptyset]$ is finite.*

Example 11. We have that **[always-M-consumed]** equals

$$\nu Z. \left([\mathbf{M-consumed}] \wedge \bigwedge_{(i,j,k) \in \mathcal{M}} (\#e_i \geq 1 \wedge \#e_j \geq 1 \Rightarrow [t_k]Z) \right),$$

where $\mathcal{M} = \{(1, 2, 1), (3, 5, 2), (6, 8, 3)\}$ (indices of pre-sets together with transition indices) and $[t]\phi = \neg\langle t \rangle \neg\phi$. That is, in order to ensure that it is impossible for a message to be preserved in any reachable state we require that **M-consumed** holds for any reachable marking.

This formula is satisfied by the Petri net underlying the Petri graph in Fig. 3 and it can even be verified using standard techniques for coverability checking.

Finally, we state correctness of the encoding. This result, together with Corollary 1 allows to check that a formula F in $\mu\mathcal{G}1$, typed as reflected, holds for a GTS \mathcal{R} by checking that its encoding $[F]$ holds in the Petri net underlying any covering of \mathcal{R} .

Proposition 5 (correct encoding). *Let $P = \langle G, N, p \rangle$ be a Petri graph and let F be a closed formula in $\mu\mathcal{G}1$. Then $m_0 \models_{\emptyset} [F, \emptyset, \emptyset, \emptyset]$ iff $P \models F$.*

7 Conclusion

We have enriched an existing approach for the verification of behavioral properties of GTSs via approximation [1, 4, 5]. The original approach approximates a GTS by a Petri graph and reduces temporal graph formulae to existing logics for Petri nets. The original logic proposed in [5] does not allow to interleave temporal modalities and edge quantifiers and is thus not able to express properties like *an edge is never removed*. We have proposed a solution to this, by using a logic that interleaves temporal and structural aspects of a GTS and extending the encoding into Petri net logics.

Our work is not the first one that proposes a non-propositional temporal logic for graph transformation systems. The most relevant approach, in this respect, is [20], where a second-order LTL logic is proposed, which is interpreted on “graph transition systems” (these however are defined in a slightly different way). Our approach is more general in the sense that also consider systems with a possibly infinite state space and approximations of these systems, whereas [20] considers finite-state systems; furthermore the temporal aspect of our logic subsumes LTL. However, a precise comparison of the two approaches is not easy, because the graph-related aspect of the logics are different ([20] considers a logic to express path properties as regular expressions, while ours is based on a fragment of the monadic second-order logic of graphs [8]), and graph items which are deleted are handled differently in the two approaches.

Another first order temporal logic—called *evolution logic*—is proposed in [26], in a framework based on abstract interpretation for the verification of Java programs featuring dynamic allocation and deallocation of objects and threads.

Evolution logic is a first order version of LTL, enriched with transitive closure, and we think it suitable to express complex properties of graph transformation systems as well: a deep comparison in this respect with our logic $\mu\mathcal{G}2$ is planned as future work.

In [22] it has been shown how the verification problem for CTL with additional quantification over “items” can be reduced to the verification of standard CTL. This is not directly applicable to our setting, since we consider infinite-state systems, but the connection to [22] deserves further study. Another related work is [9], which is concerned with the approximation of special kinds of graphs and the verification of a similar logic for verifying pointer structures on a heap.

In future work we plan to study the decidability of fragments of our logic. First, we can profit from decidability results on similar logics like the guarded monadic fragments of first-order temporal logics [16] and similar approaches for the modal μ -calculus with first-order predicates [13]. Note that from a practical point of view we can focus on the target Petri net logic \mathcal{P} . Although the full logic is undecidable, there are some clearly identifiable decidable fragments [12, 17]. In the linear-time case for instance, it is decidable to show whether there exists a run satisfying a formula containing only “eventually” operators, but mixing of “eventually” and “generally” operators in general leads to an undecidable logic.

Additionally, further approximation on the transition system generated by the Petri net can be used in order to model-check formulas on infinite-state Petri nets (see, e.g., [24]). We plan to enhance our approach by extending the encoding to the full logic including second-order quantification, and considering more general graph transformation systems, allowing non-discrete interfaces [2].

References

1. P. Baldan, A. Corradini, and B. König. A static analysis technique for graph transformation systems. In *CONCUR'01*, volume 2154 of *Lecture Notes in Computer Science*. Springer, 2001.
2. P. Baldan, A. Corradini, and B. König. Verifying finite-state graph grammars: An unfolding-based approach. In *Proc. of CONCUR'04*, volume 3170 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2004.
3. P. Baldan, A. Corradini, B. König, and B. König. Verifying a behavioural logic for graph transformation systems. In *Proc. of COMETA'03*, volume 104 of *ENTCS*, pages 5–24. Elsevier, 2004.
4. P. Baldan and B. König. Approximating the behaviour of graph transformation systems. In *Proc. of ICGT'02*, volume 2505 of *Lecture Notes in Computer Science*, pages 14–29. Springer, 2002.
5. P. Baldan, B. König, and B. König. A logic for analyzing abstractions of graph transformation systems. In *SAS'03*, volume 2694 of *Lecture Notes in Computer Science*, pages 255–272. Springer, 2003.
6. J. Bradfield and C. Stirling. Modal logics and mu-calculi: an introduction. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*. Elsevier, 2001.
7. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. *Algebraic Approaches to Graph Transformation I: Basic Concepts and Double Pushout Approach*, chapter 3. Volume 1 of Rozenberg [23], 1997.

8. B. Courcelle. *The expression of graph properties and graph transformations in monadic second-order logic*, chapter 5, pages 313–400. Volume 1 of Rozenberg [23], 1997.
9. D. Distefano, J.-P. Katoen, and A. Rensink. Who is pointing when to whom? In K. Lodaya and M. Mahajan, editors, *Proc. of FSTTCS'04*, volume 3328 of *Lecture Notes in Computer Science*, pages 250–262. Springer, 2004.
10. F. Dotti, L. Foss, L. Ribeiro, and O. Marchi Santos. Verification of distributed object-based systems. In *Proc. of FMOODS '03*, volume 2884 of *Lecture Notes in Computer Science*, pages 261–275. Springer, 2003.
11. H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. *Algebraic Approaches to Graph Transformation II: Single Pushout Approach and Comparison with Double Pushout Approach*, chapter 4. Volume 1 of Rozenberg [23], 1997.
12. J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Journal of Information Processing and Cybernetic*, 30(3):143–160, 1994.
13. E. Franconi and D. Toman. Fixpoint extensions of temporal description logics. In *Proc. of the International Workshop on Description Logics (DL2003)*, volume 81 of *CEUR Workshop Proceedings*, 2003.
14. F. Gadducci, R. Heckel, and M. Koch. A fully abstract model for graph interpreted temporal logic. In *Proc. of TAGT'98*, volume 1764 of *Lecture Notes in Computer Science*, pages 310–322. Springer, 1998.
15. M. Große-Rhode. Algebra transformation systems as a unifying framework. *Electronic Notes in Theoretical Computer Science*, 51, 2001.
16. I. Hodkinson, F. Wolter, and M. Zakharyashev. Monadic fragments of first-order temporal logics: 2000-2001 a.d. In *Proc. of LPAR'01*, pages 1–23. Springer, 2001.
17. R. R. Howell, L. E. Rosier, and H.-C. Yen. A taxonomy of fairness and temporal logic problems for Petri nets. *Theoretical Computer Science*, 82:341–372, 1991.
18. B. König and V. Kozioura. Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In *Proc. of TACAS '06*, volume 3920 of *Lecture Notes in Computer Science*, pages 197–211. Springer, 2006.
19. C. Loiseaux, S. Grafa, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–35, 1995.
20. A. Rensink. Towards model checking graph grammars. In *Proc. of the 3rd Workshop on Automated Verification of Critical Systems*, Technical Report DSSE-TR-2003-2, pages 150–160. University of Southampton, 2003.
21. A. Rensink. Canonical graph shapes. In *Proc. of ESOP '04*, volume 2986 of *Lecture Notes in Computer Science*, pages 401–415. Springer, 2004.
22. A. Rensink. Model checking quantified computation tree logic. In *Proc. of CONCUR '06*, volume 4137 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2006.
23. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation: Foundations*, volume 1. World Scientific, 1997.
24. K. Schmidt. Model-checking with coverability graphs. *Formal Methods in System Design*, 15(3), 1999.
25. D. Varró. Automated formal verification of visual modeling languages by model checking. *Software and System Modeling*, 3(2):85–113, 2004.
26. E. Yahav, T. Reps, M. Sagiv, and R. Wilhelm. Verifying temporal heap properties specified via evolution logic. In *Proc. of ESOP '03*, volume 2618 of *Lecture Notes in Computer Science*, pages 204–222. Springer, 2003.