A Logic for Analyzing Abstractions of Graph Transformation Systems*

Paolo Baldan[1], Barbara König[2], and Bernhard König[3]

[1] Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy
[2] Institut für Informatik, Technische Universität München, Germany
[3] Department of Mathematics, University of California, Irvine, USA

baldan@dsi.unive.it koenigb@in.tum.de bkoenig@math.uci.edu

Abstract. A technique for approximating the behaviour of graph transformation systems (GTSs) by means of Petri net-like structures has been recently defined in the literature. In this paper we introduce a monadic second-order logic over graphs expressive enough to characterise typical graph properties, and we show how its formulae can be effectively verified. More specifically, we provide an encoding of such graph formulae into quantifier-free formulae over Petri net markings and we characterise, via a type assignment system, a subclass of formulae F such that the validity of F over a GTS $\mathcal{G}$ is implied by the validity of the encoding of F over the Petri net approximation of $\mathcal{G}$. This allows us to reuse existing verification techniques, originally developed for Petri nets, to model-check the logic, suitably enriched with temporal operators.

1 Introduction

Distributed and mobile systems can often be specified by graph transformation systems (GTSs) in a very natural way. However, work on static analysis and verification of GTSs is scarce. The fact that GTSs can be seen as a proper extension of Petri nets suggests the possibility of relying on techniques already developed in the literature for this related formalism. However, unlike Petri nets, graph transformation systems are usually Turing-complete so that many problems decidable for general P/T-nets become undecidable for GTSs.

A technique proposed in [1, 2] is based on the approximation of GTSs by means of Petri net-like structures in the spirit of abstract interpretation of reactive systems [10]. More precisely, an approximated unfolding construction maps any given GTS $\mathcal{G}$ to a finite structure $\mathcal{U}(\mathcal{G})$, called *covering* (or approximated unfolding) of $\mathcal{G}$. The covering $\mathcal{U}(\mathcal{G})$ is a so-called *Petri graph*, i.e. a structure consisting of a Petri net with a graphical structure over places. It provides an *over-approximation* of the behaviour of $\mathcal{G}$, in the sense that any graph reachable in $\mathcal{G}$ can be mapped homomorphically to the graph underlying $\mathcal{U}(\mathcal{G})$ and its image is a reachable marking of $\mathcal{U}(\mathcal{G})$. (Note that, since $\mathcal{G}$ is possibly infinite-state,

* Research supported by the MIUR Project COFIN 2001013518 CoMeta, the FET-GC Project IST-2001-32747 Agile and the EC RTN 2-2001-00346 SegraVis.

while $\mathcal{U}(\mathcal{G})$ is finite, it would not be possible to have in $\mathcal{U}(\mathcal{G})$ isomorphic images of all graphs reachable in $\mathcal{G}$.) Therefore, given a property over graphs reflected by graph morphisms, if it holds for all states reachable in the abstraction $\mathcal{U}(\mathcal{G})$ then it also holds for all reachable graphs in $\mathcal{G}$. In other words, if T is a temporal logic formula containing only universal quantifiers (e.g. a formula in ACTL* or in a suitable fragment of the modal μ-calculus) and where state predicates are reflected by graph morphisms, then the validity of T over the covering $\mathcal{U}(\mathcal{G})$ allows us to infer the validity of T for the original system [3].

However, several relevant questions remain to be answered. First of all, which logic should we use to specify state predicates (i.e., graph properties)? How can we identify a subclass of such predicates which is reflected by graph morphisms and which can thus be safely checked over the approximation? And finally, given the approximation $\mathcal{U}(\mathcal{G})$, is there a way of encoding formulae expressing graph properties into "equivalent" formulae over Petri net markings?

As for the first point, we propose to describe state predicates, i.e., the graph properties of interest, by means of a monadic second-order logic $\mathcal{L}2$ on graphs, where quantification is allowed over (sets of) edges. (Similar logics are considered in [4].) Relevant graph properties can be expressed in $\mathcal{L}2$, e.g., the non-existence and non-adjacency of edges with specific labels, the absence of certain paths (related to security properties) or cycles (related to deadlock-freedom).

Regarding the second question, we introduce a type inference system characterising a subclass of formulae in the logic $\mathcal{L}2$ which are reflected by graph morphisms. Hence, given any formula F in such a class, if F can be proved for any reachable state of the approximation $\mathcal{U}(\mathcal{G})$ then we can deduce that F holds for any reachable graph of the original GTS $\mathcal{G}$.

Finally, given the approximation $\mathcal{U}(\mathcal{G})$, we define a constructive translation of graph formulae in $\mathcal{L}2$ into formulae over markings of the Petri net underlying the abstraction $\mathcal{U}(\mathcal{G})$. More precisely, any graph formula F is mapped to a formula $\hat{F}$ over markings such that a marking satisfies $\hat{F}$ if and only if the graph it represents satisfies F. Since the graph underlying $\mathcal{U}(\mathcal{G})$ is finite and fixed after computing the abstraction, we can perform quantifier elimination on graph formulae and, surprisingly, encode even monadic second-order logic formulae into propositional formulae on markings, containing only predicates of the form $\#s \leq c$ (the number of tokens in place s is smaller than or equal to c). We remark that the encoding for the first-order fragment of $\mathcal{L}2$ is simpler and can be defined inductively.

Altogether these results allow us to verify behavioural properties of a GTS by reusing existing model-checking techniques for Petri nets. In fact, given a formula T of a suitable temporal logic (e.g. a formula of ACTL* or of a fragment of the modal μ-calculus without $\diamond$ and negation), where state predicates are reflected by graph morphisms, then, by the construction mentioned above and using general results from abstract interpretation [10], T can be translated into a formula which can be checked over the Petri net underlying $\mathcal{U}(\mathcal{G})$. We recall that general temporal state-based logics over Petri nets, i.e., logics where basic predicates have the form $\#s \leq c$, are not decidable in general, but important fragments of such logics are [8, 7, 9].

For the sake of simplicity, although the approximation method of [1, 2] was originally designed for hypergraphs, in this paper we concentrate on directed graphs. The extension to general hypergraphs requires some changes to the graph logic $\mathcal{L}2$. This rises some technical difficulties which are, while not being insurmountable, a hindrance to the clear and easy presentation of our results.

In the rest of the paper we will first summarise the approximation technique for GTSs in [1], shortly mentioning some results from [2]. Then, we will define the monadic second-order logic $\mathcal{L}2$ over graphs and we will introduce the type system characterising a subclass of formulae in $\mathcal{L}2$ which are reflected by graph morphisms, and which can thus be checked on the covering. Finally we will show how to encode these formulae into quantifier-free state-based formulae on the markings of Petri nets, starting from the simpler case of first-order formulae.

2 Approximated Unfolding Construction

In this section we sketch the algorithm, introduced in [1], for the construction of a finite approximation of the unfolding of a graph transformation system. We first define graphs and structure-preserving morphisms on graphs. We will assume that Λ denotes a fixed and finite set of labels. Note that multiple edges between nodes are allowed.

Definition 1 (Graph, graph morphism). A *graph* $G = (V_G, E_G, s_G, t_G, l_G)$ consists of a set V_G of nodes, a set E_G of edges, a source and a target function $s_G, t_G \colon E_G \to V_G$ and a function $l_G \colon E_G \to \Lambda$ labelling the edges.

A *graph morphism* $\varphi \colon G_1 \to G_2$ is a pair of mappings $\varphi_V \colon V_{G_1} \to V_{G_2}$ and $\varphi_E \colon E_{G_1} \to E_{G_2}$ such that $\varphi_V \circ s_{G_1} = s_{G_2} \circ \varphi_E$, $\varphi_V \circ t_{G_1} = t_{G_2} \circ \varphi_E$ and $l_{G_1} = l_{G_2} \circ \varphi_E$ for each edge $e \in E_{G_1}$. A morphism φ will be called *edge-bijective* if φ_E is a bijection. The subscripts in φ_E and φ_V will be usually omitted.

We next define the notion of a graph transformation system and the corresponding rewriting relation.

Definition 2 (Graph transformation system). A *graph transformation system (GTS)* $(G_0, \mathcal{R})$ consists of an initial graph G_0 and a set $\mathcal{R}$ of rewriting rules of the form $r = (L, R, \alpha)$, where L, R are graphs, called *left-hand side* and *right-hand side*, respectively, and $\alpha \colon V_L \to V_R$ is an injective function.

A *match* of a rewriting rule r in a graph G is a morphism $\varphi \colon L \to G$ which is injective on edges. We can apply r to a match in G obtaining a new graph H, written $G \overset{r}{\Rightarrow} H$. The target graph H is defined as follows

$$V_H = V_G \uplus (V_R - \alpha(V_L)) \qquad E_H = (E_G - \varphi(E_L)) \uplus E_R$$

and, defining $\overline{\varphi} : V_R \to V_H$ by $\overline{\varphi}(\alpha(v)) = \varphi(v)$ if $v \in V_L$ and $\overline{\varphi}(v) = v$ otherwise, the source, target and labelling functions are given by

$$\begin{array}{rcl} e \in E_G - \varphi(E_L) & \Rightarrow & s_H(e) = s_G(e), \quad t_H(e) = t_G(e), \quad l_H(e) = l_G(e) \\ e \in E_R & \Rightarrow & s_H(e) = \overline{\varphi}(s_R(e)), \quad t_H(e) = \overline{\varphi}(t_R(e)), \quad l_H(e) = l_R(e) \end{array}$$

Intuitively, the application of r to G at the match φ first removes from G the image of the edges of L. Then the graph G is extended by adding the new nodes in R (i.e., the nodes in $V_R - \alpha(V_L)$) and the edges of R. Observe that the (images of) the nodes in L are preserved, i.e., not affected by the rewriting step.

Example 1. Consider a system where processes compete for resources R_1 and R_2. A process needs both resources in order to perform some task. The system is represented as a GTS Sys as follows. We consider edges labelled by R_1, R_2, R_1^f, R_2^f standing for assigned and free resources, respectively, and P_1, P_2 and P_3 denoting a process waiting for resource R_1, a process waiting for resource R_2 and a process holding both resources, respectively. Furthermore, edges labelled by D_1 and D_2 connect the target node of a process and the source node of a resource when the process is asking for the resource. When the target node of a resource coincides with the source node of a process, this means that the resource is assigned to the process. The initial scenario for Sys is represented in Fig. 1, with a single process P_1 asking for both resources.

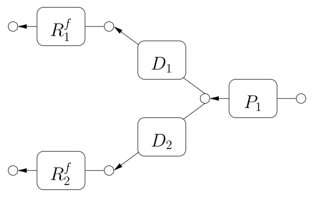

Fig. 1. Start graph of Sys with a process and resources.

The rewriting rules of Sys are defined with the aim of avoiding deadlocks in the form of vicious cycles. There are three kind of rules, depicted in Fig. 2: (1) a process P_i can acquire a free resource R_j^f whenever $i = j$ and become P_{i+1}, (2) P_3 can release its resources and (3) processes of the form P_1 can fork creating more processes of the same kind with demand for the same resources. The natural numbers $1, 2, 3, \ldots$ which decorate nodes in the left-hand side and right-hand side of rules implicitly represent the mapping α.

Observe that an additional rule, analogous to rule 1, but with $i = 1$ and $j = 2$, would possibly lead to a vicious cycle with circular demand for resources, in two steps (see Fig. 3).

Some basic notation concerning multisets is needed to deal with Petri nets. 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 sometime identify $A^\oplus$ with the set of functions $m \colon A \to \mathbb{N}$ 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 also identified with the underlying set, writing, e.g., $a \in m$ for $m(a) \neq 0$. Given a function $f \colon A \to B$, by $f^\oplus \colon A^\oplus \to B^\oplus$ we denote its monoidal extension, i.e., $f^\oplus(m)(b) = \sum_{f(a)=b} m(a)$ for every $b \in B$.

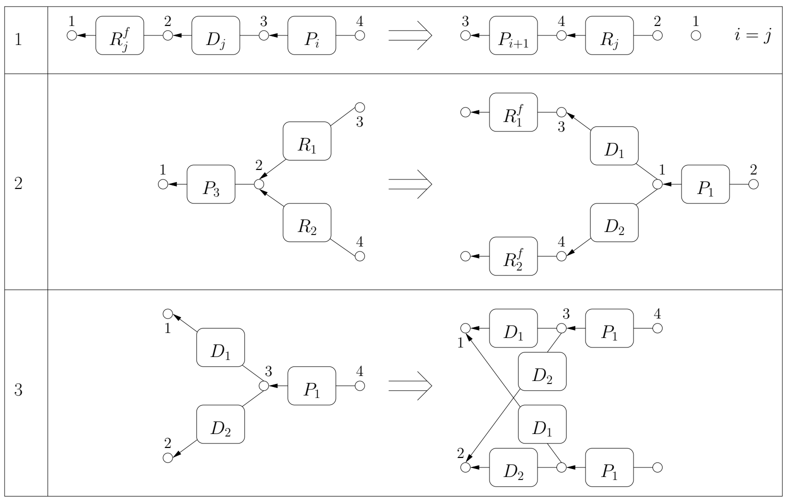

Fig. 2. Rewriting rules of the GTS Sys.

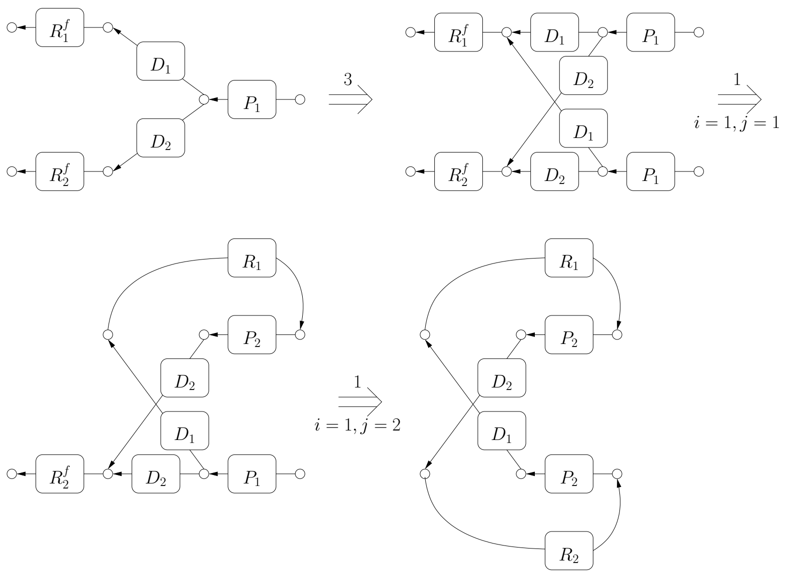

Fig. 3. Vicious cycle representing a deadlock.

In order to approximate graph transformation systems we use Petri graphs, introduced in [1], which are basically Petri nets, specifying the operational behaviour, with added graph structure.

Definition 3 (Petri graphs). Let $\mathcal{G} = (G_0, \mathcal{R})$ be a GTS. A *Petri graph P (over $\mathcal{G}$)* is a tuple (G, N, m_0) where

– G is a graph;
– $N = (E_G, T_N, {}^\bullet(), ()^\bullet, p_N)$ is a Petri net, where the set of places E_G is the edge set, T_N is the set of transitions, ${}^\bullet(), ()^\bullet : T_N \to E_G^\oplus$ specify the post-set and pre-set of each transition and $p_N : T_N \to \mathcal{R}$ is the labelling function;
– $m_0 \in (E_G)^\oplus$ is the *initial marking* of the Petri graph, satisfying $m_0 = \iota^\oplus(E_{G_0})$ for a suitable graph morphism $\iota : G_0 \to G$ (i.e., m_0 must properly correspond to the initial state of the GTS $\mathcal{G}$).

A marking $m \in E_G^\oplus$ will be called *reachable (coverable)* in P if it is reachable (coverable) from the initial marking in the Petri net underlying P.

Remark. The definition of Petri graph is slightly different from the original one in [1], in that we omit some graph morphisms associated to transitions (the μ-component) and to the initial marking, and the so-called irredundancy condition. Both are needed for the actual construction of the Petri graph from a GTS, but they play no role in the results of this paper.

A marking m of a Petri graph can be seen as an abstract representation of a graph in the following sense.

Definition 4. Let (G, N, m_0) be a Petri graph and let $m \in E_G^\oplus$ be a marking of N. The graph *generated* by m, denoted by *graph*(m), is the graph H defined as follows: $V_H = \{v \in V_G \mid \exists e \in m : (s_G(e) = v \vee t_G(e) = v)\}$, $E_H = \{(e, i) \mid e \in m \wedge 1 \leq i \leq m(e)\}$, $s_H((e, i)) = s_G(e)$, $t_H((e, i)) = t_G(e)$ and $l_H((e, i)) = l_G(e)$.

Alternatively the graph *graph*(m) can be defined as the unique graph H, up to isomorphism, such that there exists a morphism $\psi : H \to G$ injective on nodes with $\psi^\oplus(E_H) = m$. An example of a Petri net marking with the corresponding generated graph can be found in Fig. 4.

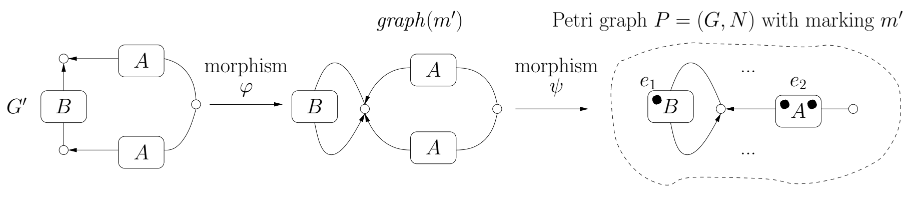

Fig. 4. A pair (G', m') contained in a simulation.

Given a GTS $(G_0, \mathcal{R})$, with some minor constraints on the format of rewriting rules (see [1, 2]), we can construct a Petri graph approximation of $(G_0, \mathcal{R})$, called *covering* and denoted by $\mathcal{U}(G_0, \mathcal{R})$. The covering is produced by the last step of the following (terminating) algorithm which generates a sequence $P_i = (G_i, N_i, m_i)$ of Petri graphs.

1. $P_0 = (G_0, N_0, m_0)$, where the net N_0 contains no transitions and $m_0 = E_{G_0}$.
2. As long as one of the following steps is applicable, transform P_i into P_{i+1}, giving precedence to folding steps.

   Unfolding. Find a rule $r = (L, R, \alpha) \in \mathcal{R}$ and a match $\varphi\colon L \to G_i$ such that $\varphi(E_L^{\oplus})$ is coverable in P_i. Then extend P_i by "attaching" R to G_i according to α and add a transition t, labelled by r, describing the application of rule r.

   Folding. Find a rule $r = (L, R, \alpha) \in \mathcal{R}$ and two matches $\varphi, \varphi'\colon L \to G_i$ such that $\varphi^{\oplus}(E_L)$ and $\varphi'^{\oplus}(E_L)$ are coverable in N_i and the second match is causally dependent on the transition unfolding the first match. Then merge the two matches by setting $\varphi(e) \equiv \varphi'(e)$ for each $e \in E_L$ and factoring through the resulting equivalence relation $\equiv$.

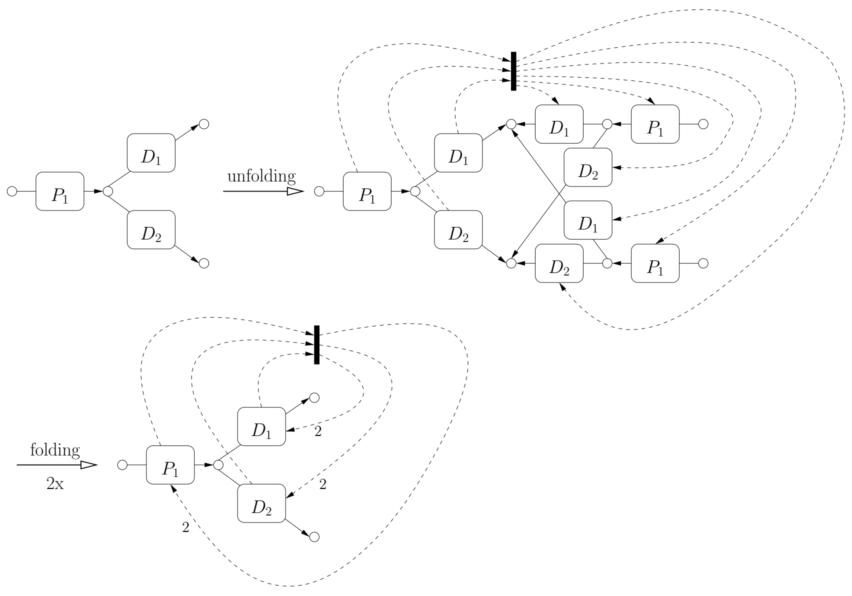

Fig. 5. An unfolding and two folding steps.

For instance an unfolding step involving rule 3 is depicted in Fig. 5. Transitions are represented as black rectangles and the Petri net structure is rendered by connecting edges (places) to transitions with dashed lines. The label k for dashed lines represents the weight with which the target/source place occurs in the post-set (pre-set); when the weight is 1, the label is omitted. In the resulting

Petri graph we can find three occurrences of the left-hand side of rule 3. The latter two are causally dependent on the first, which means that they can be merged in two folding steps. The algorithm, starting from the start graph in Fig. 1, terminates producing the Petri graph $\mathcal{U}(\mathsf{Sys})$ in Fig. 6, where the initial marking is represented by tokens.

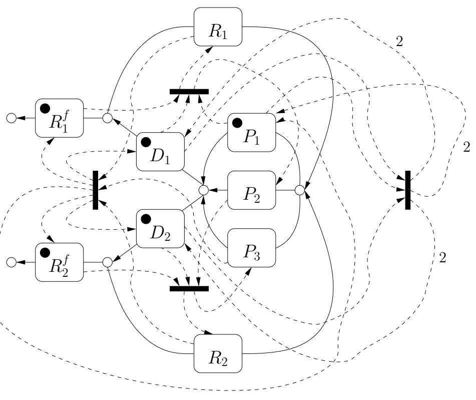

Fig. 6. The Petri graph $\mathcal{U}(\mathsf{Sys})$ computed as covering of Sys.

The covering $\mathcal{U}(G_0, \mathcal{R})$ is an abstraction of the original GTS $(G_0, \mathcal{R})$ in the following sense.

Proposition 1 (Abstraction). *Let $\mathcal{G} = (G_0, \mathcal{R})$ be a graph transformation system and let $\mathcal{U}(\mathcal{G}) = (G, N, m_0)$ be its covering. Furthermore let $\mathbf{G}$ be the set of graphs reachable from G_0 in $\mathcal{G}$ and let $\mathbf{M}$ be the set of reachable markings in $\mathcal{U}(\mathcal{G})$. Then there exists a simulation $S \subseteq \mathbf{G} \times \mathbf{M}$ with the following properties:*

- *$(G_0, m_0) \in S$;*
- *whenever $(G', m') \in S$ and $G' \overset{r}{\Rightarrow} G''$, then there exists a marking m'' with $m' \xrightarrow{r} m''$ and $(G'', m'') \in S$;*
- *for every $(G', m') \in S$ there is an edge-bijective morphism $\varphi: G' \to graph(m')$.*

The above result will allow us to use existing results concerning abstractions of reactive systems [3, 10]. Consider the system Sys in our running example. We would like to verify that, according to the design intentions, Sys is deadlock-free. This is formalised by the requirement that all reachable graphs do not contain a vicious cycle, i.e., a cycle of edges where P_2-labelled edges (processes holding a resource and waiting for a second resource) occur twice. This graph property is reflected by graph morphisms, hence, by using Proposition 1, if we can prove it on the covering $\mathcal{U}(\mathsf{Sys})$, we could deduce that it holds for the original system Sys as well. Observe that actually, in this case, even the stronger property $\#e \leq 1$, where e is the edge labelled P_2, holds for all reachable markings as it can be easily verified by drawing the coverability graph of the Petri net. This is an ad hoc proof of the property, which instead, by the results in this paper, will follow as an instance of a general theory.

The idea that will be concretized by the results in the paper, is the following. Let $\mathcal{G}$ be a GTS and let $\mathcal{U}(\mathcal{G})$ be its covering. By Proposition 1, $\mathcal{U}(\mathcal{G}) = (G, N, m_0)$ "approximates" $\mathcal{G}$ via a simulation consisting of pairs (G', m') such that G' can be mapped to *graph*(m') (see, e.g., Fig. 4) via an edge-bijective morphism. Given a formula on graphs F, expressing a state property in $\mathcal{G}$, a corresponding formula $M(F)$ on the markings of $\mathcal{U}(\mathcal{G})$ is constructed such that, for any pair in the simulation,

$$m' \models M(F) \;\Rightarrow\; G' \models F.$$

This will be obtained in two steps. First, we will identify formulae F which are reflected by edge-bijective morphisms, ensuring that *graph*$(m') \models F$ implies $G' \models F$. Then, we will encode F into a propositional formula $M(F)$ on multisets such that $m' \models M(F) \iff \mathit{graph}(m') \models F$.

Call $\mathcal{F}$ the above mentioned class of graph formulae. Now, one can consider a temporal logic over GTSs, where basic predicates are taken from $\mathcal{F}$. For suitable fragments of such logics, e.g., the modal μ-calculus without negation and the "possibility operator" $\Diamond$, by Proposition 1 and exploiting general results in [10], any temporal formula T over graphs can be translated to a formula $M(T)$ over markings (translating the basic predicates as above), such that, if $N \models M(T)$ then $\mathcal{G} \models T$, i.e., T is valid for the original GTS.

3 A Second-Order Monadic Logic for Graphs

We introduce the monadic second-order logic $\mathcal{L}2$ for specifying graph properties. Quantification is allowed over edges, but not over nodes (as, e.g., in [4]).

Definition 5 (Graph formula). Let $\mathcal{X}_1 = \{x, y, z, \ldots\}$ be a set of (first-order) edge variables and let $\mathcal{X}_2 = \{X, Y, Z, \ldots\}$ be a set of (second-order) variables representing edge sets. The set of *graph formulae* of the logic $\mathcal{L}2$ is defined as follows, where $\ell \in \Lambda$

$$\begin{array}{lll} F \;::= & x = y \;\mid\; s(x) = s(y) \;\mid\; s(x) = t(y) \;\mid\; t(x) = t(y) \;\mid & \\ & \mathit{lab}(x) = \ell \;\mid\; x \in X & \text{(Predicates)} \\ & F \vee F \;\mid\; F \wedge F \;\mid\; F \Rightarrow F \;\mid\; \neg F & \text{(Connectives)} \\ & \forall x.F \;\mid\; \exists x.F \;\mid\; \forall X.F \;\mid\; \exists X.F & \text{(Quantifiers)} \end{array}$$

We denote by *free*(F) and *Free*(F) the sets of first-order and second-order variables, respectively, occurring free in F, defined in the obvious way.

Note that, even if quantification over nodes is disallowed, formulae expressing properties of classes of nodes can be easily stated, e.g., the property "for all non-isolated nodes v it holds that $P(v)$" is formalised as "$\forall x.(P(s(x)) \wedge P(t(x)))$".

Definition 6 (Quantifier depth). The first-order and second-order *quantifier depth* ($\mathrm{qd}_1(F)$ and $\mathrm{qd}_2(F)$, respectively) of a graph formula F in $\mathcal{L}2$ is inductively defined as follows, where A is a predicate, $\mathit{op} \in \{\wedge, \vee, \Rightarrow\}$ and $i \in \{1, 2\}$.

$$\mathrm{qd}_i(A) = 0 \qquad \mathrm{qd}_i(\neg F_1) = \mathrm{qd}_i(F_1) \qquad \mathrm{qd}_i(F_1\, \mathit{op}\, F_2) = \max\{\mathrm{qd}_i(F_1), \mathrm{qd}_i(F_2)\}$$
$$\mathrm{qd}_1(\forall x.F_1) = \mathrm{qd}_1(\exists x.F_1) = \mathrm{qd}_1(F_1) + 1 \qquad \mathrm{qd}_2(\forall x.F_1) = \mathrm{qd}_2(\exists x.F_1) = \mathrm{qd}_2(F_1)$$
$$\mathrm{qd}_1(\forall X.F_1) = \mathrm{qd}_1(\exists X.F_1) = \mathrm{qd}_1(F_1) \quad \mathrm{qd}_2(\forall X.F_1) = \mathrm{qd}_2(\exists X.F_1) = \mathrm{qd}_2(F_1) + 1$$

The notion of satisfaction is defined in a straightforward way.

Definition 7 (Satisfaction). Let G be a graph, let F be a graph formula in $\mathcal{L}2$, let $\sigma : \mathit{free}(F) \to E_G$ and $\Sigma : \mathit{Free}(F) \to \mathcal{P}(E_G)$ be valuations for the free first- and second-order variables of F, respectively. The *satisfaction relation* $G \models_{\sigma,\Sigma} F$ is defined inductively, in the usual way; for instance:

$$\begin{aligned} G \models_{\sigma,\Sigma} x = y &\iff \sigma(x) = \sigma(y) \\ G \models_{\sigma,\Sigma} s(x) = s(y) &\iff s_G(\sigma(x)) = s_G(\sigma(y)) \\ G \models_{\sigma,\Sigma} \mathit{lab}(x) = \ell &\iff l_G(\sigma(x)) = \ell \\ G \models_{\sigma,\Sigma} x \in X &\iff \sigma(x) \in \Sigma(X) \end{aligned}$$

Example 2. The formula NC_ℓ below states that a graph does not contain a cycle including two distinct edges labelled ℓ, a property that will be used to express the absence of vicious cycles in our system Sys. It is based on the formula $\mathit{NP}(x, y)$, which says that there is no path connecting the edges x and y, stating that a set that contains at least all successors of x does not always contain y.

$$\begin{aligned} \mathit{NP}(x, y) = \neg\forall X.(&\forall z.(t(x) = s(z) \vee \exists w.(w \in X \wedge t(w) = s(z))) \Rightarrow z \in X) \\ &\Rightarrow y \in X) \end{aligned}$$
$$\mathit{NC}_\ell = \forall x.\forall y.(\mathit{lab}(x) = \ell \wedge \mathit{lab}(y) = \ell \wedge \neg(x = y) \Rightarrow \mathit{NP}(x, y) \vee \mathit{NP}(y, x))$$

The following standard argument shows that this property can not be stated in first-order logic, a fact which motivates our choice of considering a second-order logic: it is easy to find sentences ψ_n in first-order logic stating that ‘there is no cycle of length $\leq n$ through two distinct edges labelled ℓ’. Every finite subset of the theory $T = \{\neg \mathit{NC}_\ell\} \cup \{\psi_n\}_{n \in \mathbb{N}}$ is satisfiable but T itself is not satisfiable. The compactness theorem rules this out for first-order theories, so NC_ℓ cannot be first-order.

4 Preservation and Reflection of Graph Formulae

In this section we introduce a type system over graph formulae in $\mathcal{L}2$ which allows us to single out subclasses of formulae preserved or reflected by edge-bijective morphisms. By Proposition 1, given a GTS $\mathcal{G}$ every graph reachable in $\mathcal{G}$ can be mapped homomorphically via an edge-bijective morphism to the graph generated by a marking reachable in the covering $\mathcal{U}(\mathcal{G})$ of $\mathcal{G}$. Hence a formula reflected by all edge-bijective morphisms can be safely checked over the approximation $\mathcal{U}(\mathcal{G})$, in the sense that if it holds in $\mathcal{U}(\mathcal{G})$, then we can deduce that it holds also in $\mathcal{G}$.

To define the notions of reflection (and preservation) of general graph formulae, possibly with free variables, observe that valuations are naturally "transformed" under graph morphisms. Let F be formula, let $\varphi : G_1 \to G_2$ be a graph morphism, and let $\sigma_1 : \mathit{free}(F) \to E_{G_1}$ and $\Sigma_1 : \mathit{Free}(F) \to \mathcal{P}(E_{G_1})$ be valuations. A valuation for the first-order variables of F in G_2 is naturally given by $\varphi \circ \sigma_1$, while a valuation Σ_2 for second-order variables can be defined by $\Sigma_2(X) = \varphi(\Sigma_1(X))$ for any variable X. Abusing the notation, Σ_2 will be denoted by $\varphi \circ \Sigma_1$.

Definition 8 (Reflection and Preservation). Let F be a formula in $\mathcal{L}2$ and let $\varphi\colon G_1 \to G_2$ be a graph morphism. We say that F is *preserved by* φ if for all valuations $\sigma_1\colon \mathit{free}(F) \to E_{G_1}$ and $\Sigma_1\colon \mathit{Free}(F) \to \mathcal{P}(E_{G_1})$

$$G_1 \models_{\sigma_1,\Sigma_1} F \quad \Rightarrow \quad G_2 \models_{\varphi\circ\sigma,\varphi\circ\Sigma_1} F.$$

Symmetrically, F is *reflected* by φ if the above holds where $\Rightarrow$ is replaced by $\Leftarrow$.

Observe that, in particular, a closed formula F is preserved by a graph morphism $\varphi : G_1 \to G_2$ if $G_1 \models_{\emptyset,\emptyset} F$ implies $G_2 \models_{\emptyset,\emptyset} F$.

As mentioned above we are interested in syntactic criteria characterising classes of graph formulae reflected, respectively preserved, by all edge-bijective graph morphisms. For first-order predicate logic, criteria for arbitrary morphisms can be found in [6]. Here we provide a technique which works for general second-order monadic formulae, based on a type system assigning to every formula F either $\rightarrow$, meaning that F is preserved, or $\leftarrow$, meaning that F is reflected by edge-bijective morphisms. The type rules are given in Fig. 7 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 $F_2 : d$.

Typing predicates:

$$s(x) = s(y),\ s(x) = t(y),\ t(x) = t(y)\colon \rightarrow \qquad x = y,\ \mathit{lab}(x) = \ell,\ x \in X\colon \leftrightarrow$$

Typing connectives and quantifiers:

$$\frac{F\colon d}{\neg F\colon d^{-1}} \qquad \frac{F_1, F_2\colon d}{F_1 \vee F_2, F_1 \wedge F_2\colon d} \qquad \frac{F_1\colon d^{-1}, \quad F_2\colon d}{F_1 \Rightarrow F_2\colon d} \qquad \frac{F\colon d}{\forall x.F\colon d} \qquad \frac{F\colon d}{\exists x.F\colon d}$$

$$\frac{F\colon d}{\forall X.F\colon d} \qquad \frac{F\colon d}{\exists X.F\colon d}$$

Fig. 7. The type system for preservation and reflection.

The type system can be shown to be correct.

Proposition 2 (Correctness). *Let F be a graph formula. If $F\colon\rightarrow$ is provable then F is preserved by all edge-bijective morphisms. Similarly, if $F\colon\leftarrow$ is provable then F is reflected by all edge-bijective graph morphisms.*

Example 3. It holds that $NP(x,y)\colon\leftarrow$ and $NC_\ell\colon\leftarrow$, i.e., absence of paths and of vicious cycles is reflected by edge-bijective morphisms.

Not all formulae that are preserved respectively reflected are recognised by the above type system. The following result shows that this incompleteness is a fundamental problem, due to the undecidability of reflection and preservation.

Proposition 3 (Undecidability of the Reflection (Preservation) Problem for formulae). *The following two sets are undecidable:*

$$\begin{aligned} \mathit{Refl}_{FO} &= \{F \mid F \textit{ closed first-order formula, reflected by edge-bijective} \\ &\qquad \textit{graph morphisms}\} \\ \mathit{Pres}_{FO} &= \{F \mid F \textit{ closed first-order formula, preserved by edge-bijective} \\ &\qquad \textit{graph morphisms}\} \end{aligned}$$

5 A Propositional Logic on Multisets

In order to characterise markings of Petri nets we use the following logic on multisets. We consider a fixed universe A over which all multisets are formed.

Definition 9 (Multiset formula). The set of *multiset formulae*, ranged over by M, is defined as follows, where $a \in A$ and $c \in \mathbb{N}$

$$M \ ::= \ \#a \leq c \ \mid \ \neg M \ \mid \ M \vee M' \ \mid \ M \wedge M'.$$

Let m be a multiset with elements from A. The satisfaction relation $m \models M$ is defined, on basic predicates, as $m \models (\#a \leq c) \iff m(a) \leq c$. Logical connectives are dealt with as usual.

We will consider also derived predicates of the form $\#a \geq c$ and $\#a = c$ where

$$(\#a \geq c) = \begin{cases} \neg(\#e \leq c-1) & \text{if } c > 0 \\ \mathit{true} & \text{otherwise} \end{cases}, \qquad (\#e = c) = (\#e \leq c) \wedge (\#e \geq c).$$

6 Encoding First-Order Graph Logic

In this section we show how first-order graph formulae can be encoded into "equivalent" multiset formulae. More precisely, given the fixed Petri graph $P = (G, N, m_0)$ the aim is to find an encoding M_1 of first-order graph formulae into multiset formulae such that $\mathit{graph}(m) \models F \iff m \models M_1(F)$ for every marking m of P and every closed first order graph formula F.

The encoding M_1 is based on the following observation: every graph $\mathit{graph}(m)$ 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 might be multiplied, generating several parallel copies of the same template edge. Whenever a formula has two free variables x, y and $\mathit{graph}(m)$ has n parallel copies $e_1, \dots, e_n$ of the same edge, it is not necessary to associate x and y with all edges, but it is sufficient to assign e_1 to x and e_2 to y (first alternative) or e_1 to both x and y (second alternative). Thus, whenever we encode a formula F, we have to keep track of the following information: a partition P on the free variables $\mathit{free}(F)$, telling us which variables are mapped to the same edge, and a mapping ρ from $\mathit{free}(F)$ to the edges of G, with $\rho(x) = e$ meaning that x will be instantiated with a copy of the template edge e. Since there might be several different copies of the same template edge, two variables x and y in different sets of P can be mapped by ρ to the same edge of G. Whenever we encode an existential quantifier $\exists x$, we have to 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, in this case x and y should be in the same set of the partition P. Or x is instantiated with a new copy of an edge in G. In this case, a new set $\{x\}$ is added to P and we have to make sure that enough edges are available by adding a suitable predicate.

We need the following notation. We will describe an equivalence relation on a set A by a partition $P \subseteq \mathcal{P}(A)$ of A, where every element of P represents an equivalence class. We will write $x\,P\,y$ whenever x, y are in the same equivalence class. Furthermore we assume that each equivalence P is associated with a function $\mathit{rep} : P \to A$ which assigns a representative to every equivalence class. The encoding given below is independent of any specific choice of representatives.

Given a function $f : A \to B$ such that $f(a) = f(a')$ for all $a, a' \in A$ with aPa' and a fixed $b \in B$ we define $n_{P,f}(b) = |\{k \in P \mid f(\mathit{rep}(k)) = b\}|$, i.e., $n_{P,f}(b)$ is the number of sets in the partition P that are mapped to b.

Definition 10. Let G be a directed graph, let F be graph formula in the first-order fragment of $\mathcal{L}2$, let $\rho : \mathit{free}(F) \to E_G$ and let $P \subseteq \mathcal{P}(\mathit{free}(F))$ be an equivalence relation such that $x\,P\,y$ implies $\rho(x) = \rho(y)$ for all $x, y \in \mathit{free}(F)$. The *encoding* M_1 is defined as follows:

$$
\begin{aligned}
M_1[\neg F, \rho, P] &= \neg M_1[F, \rho, P] \\
M_1[F_1 \vee F_2, \rho, P] &= M_1[F_1, \rho, P] \vee M_1[F_1, \rho, P] \\
M_1[F_1 \wedge F_2, \rho, P] &= M_1[F_1, \rho, P] \wedge M_1[F_1, \rho, P] \\
M_1[x = y, \rho, P] &= \begin{cases} \mathit{true} & \text{if } x\,P\,y \\ \mathit{false} & \text{otherwise} \end{cases} \\
M_1[\mathit{lab}(x) = \ell, \rho, P] &= \begin{cases} \mathit{true} & \text{if } l_G(\rho(x)) = \ell \\ \mathit{false} & \text{otherwise} \end{cases} \\
M_1[s(x) = s(y), \rho, P] &= \begin{cases} \mathit{true} & \text{if } s_G(\rho(x)) = s_G(\rho(y)) \\ \mathit{false} & \text{otherwise} \end{cases} \\
&\ \text{the formulae } t(x) = t(y) \text{ and } s(x) = t(y) \\
&\ \text{are treated analogously}
\end{aligned}
$$

$$
\begin{aligned}
M_1[\exists x.F,\rho,P] = {} & \bigvee_{k\in P}(M_1[F,\rho\cup\{x\mapsto\rho(rep(k))\},P\backslash\{k\}\cup\{k\cup\{x\}\}])\vee \\
& \bigvee_{e\in E_G}(M_1[F,\rho\cup\{x\mapsto e\},P\cup\{\{x\}\}]\wedge(\#e\geq n_{P,\rho}(e)+1)) \\
M_1[\forall x.F,\rho,P] = {} & \bigwedge_{k\in P}(M_1[F,\rho\cup\{x\mapsto\rho(rep(k))\},P\backslash\{k\}\cup\{k\cup\{x\}\}])\wedge \\
& \bigwedge_{e\in E_G}((\#e\geq n_{P,\rho}(e)+1)\Rightarrow M_1[F,\rho\cup\{x\mapsto e\},P\cup\{\{x\}\}])
\end{aligned}
$$

If F is closed formula (i.e., without free variables), we define $M_1(F) = M_1[F,\emptyset,\emptyset]$.

It is worth remarking that such an approach is similar to the model-theoretic method of quantifier elimination, defined by Tarski in the 1950's to show decidability and completeness for theories like dense linear orderings or algebraically closed fields (see [14]). We remark that here finiteness of graphs is essential.

We can now show that the encoding is correct in the sense explained above. We will omit the index Σ in $\models_{\sigma,\Sigma}$ when talking about first-order formulae only.

Proposition 4. *Let (G,N,m_0) be a Petri graph, F a first-order formula in $\mathcal{L}2$ and m a marking of N. Then it holds that*

$$graph(m)\models_\sigma F \iff m\models M_1[F,\rho,P],$$

when

- $\rho : free(F)\to E_G$;
- *P is an equivalence on $free(F)$ such that $x\,P\,y$ implies $\rho(x)=\rho(y)$ for any $x,y\in free(F)$;*
- *$\sigma : free(F)\to E_{graph(m)}$ satisfies $x\,P\,y \iff \sigma(x)=\sigma(y)$ and $\varphi\circ\sigma=\rho$, where $\varphi\colon graph(m)\to G$ denotes the projection of $graph(m)$ over G, i.e., a graph morphism such that $\varphi((e,i))=e\in E_G$.*

Whenever F is closed the proposition above trivially gives us the expected result. i.e., $graph(m)\models F$ iff $m\models M_1(F)$.

Example 4. Consider the formula $F = \exists x.(\underbrace{lab(x)=A\wedge\overbrace{\forall y.\neg(t(x)=s(y))}^{F_2}}_{F_1})$.

The graph under consideration is the graph G on the right in Fig. 4 (containing a looping B-edge e_1 and an A-edge e_2). The encoding goes as follows (with some simplifications of the formula along the way):

$$
\begin{aligned}
& M_1[F,\emptyset,\emptyset] \\
= {} & (M_1[F_1,\{x\mapsto e_1\},\{\{x\}\}]\wedge(\#e_1\geq 1))\vee(M_1[F_1,\{x\mapsto e_2\},\{\{x\}\}]\wedge(\#e_2\geq 1)) \\
= {} & (\underbrace{M_1[lab(x)=A,\{x\mapsto e_1\},\{\{x\}\}]}_{=false}\wedge M_1[F_2,\{x\mapsto e_1\},\{\{x\}\}]\wedge(\#e_1\geq 1))\vee
\end{aligned}
$$

$$\begin{aligned}
&(\underbrace{M_1[lab(x) = A, \{x \mapsto e_2\}, \{\{x\}\}]}_{=true} \wedge M_1[F_2, \{x \mapsto e_2\}, \{\{x\}\}] \wedge (\#e_2 \geq 1)) \\
\equiv\ & \underbrace{M_1[\neg(t(x) = s(y)), \{x, y \mapsto e_2\}, \{\{x, y\}\}]}_{=true} \wedge \\
& (\#e_1 \geq 1 \Rightarrow \underbrace{M_1[\neg(t(x) = s(y)), \{x \mapsto e_2, y \mapsto e_1\}, \{\{x\}, \{y\}\}]}_{=false}) \wedge \\
& (\#e_2 \geq 2 \Rightarrow \underbrace{M_1[\neg(t(x) = s(y)), \{x, y \mapsto e_2\}, \{\{x\}, \{y\}\}]}_{=true}) \wedge (\#e_2 \geq 1) \\
\equiv\ & \neg(\#e_1 \geq 1) \wedge (\#e_2 \geq 1)
\end{aligned}$$

7 Encoding Monadic Second-Order Graph Logic

In this section we show that also general monadic second-order graph formulae in $\mathcal{L}2$ can be encoded into multiset formulae. Differently from the first-order case, the encoding is not defined inductively, but, still, quantifier elimination is possible. We start with an easy but useful lemma.

Lemma 1 (Edge Permutations). *Let σ, Σ be valuations such that $G \models_{\sigma,\Sigma} F$. Furthermore let $\pi : G \to G$ be an automorphism such that $s_G(e) = s_G(\pi(e))$ and $t_G(e) = t_G(\pi(e))$. Then $G \models_{\pi\circ\sigma,\pi\circ\Sigma} F$.*

The encoding uses the fact that multiple copies of an edge are distinguished only by their identity, but have the same source and target nodes and the same label. Hence whenever we want to encode a first-order quantifier, we only have to check all the edges that have already appeared so far and a fresh copy of every edge in G. From this, as we will see, one can infer that for checking the validity of a formula F it is sufficient to consider only up to $\mathrm{qd}_1(F) \cdot 2^{\mathrm{qd}_2(F)}$ copies of every edge in the template graph G.

The following proposition basically states that if there are enough parallel edges which belong to the same sets of the form $\Sigma(X)$, where Σ is a second-order valuation and X a second-order variable, then one of these edges can be removed—provided that it is not in the range of the first-order valuation σ—without changing the validity of a formula F.

Proposition 5. *Let G be a graph, F a graph formula in $\mathcal{L}2$, let σ, Σ be valuations for the free variables in F and let $e \in E_G$ be a fixed edge. Assume that*

(1) the edge e is not in the range of σ and
(2) $|E^G_\Sigma(e)| > (\mathrm{qd}_1(F) + |\mathrm{dom}(\sigma)|) \cdot 2^{\mathrm{qd}_2(F)}$ where

$$\begin{aligned}
E^G_\Sigma(e) = \{e' \in E_G \mid\ & s_G(e) = s_G(e'), t_G(e) = t_G(e'), l_G(e) = l_G(e'), \\
& \forall X \in \mathrm{dom}(\Sigma).(e \in \Sigma(X) \iff e' \in \Sigma(X))\}
\end{aligned}$$

Then $G \models_{\sigma,\Sigma} F \iff G \setminus \{e\} \models_{\sigma,\Sigma_e} F$, where $G \setminus \{e\}$ is obtained by removing the edge e from graph G and $\Sigma_e(X) = \Sigma(X) - \{e\}$.

From Proposition 5 we infer the following corollary.

Corollary 1. *Let F be a closed graph formula in $\mathcal{L}2$. Let furthermore G be a graph and $m \in E_G^\oplus$ be a multiset over (the set of edges of) G. Then $graph(m) \models F$ if and only if $graph(m') \models F$, where $m' \in E_G^\oplus$ is defined by $m'(e) = \min\{m(e), \mathrm{qd}_1(F) \cdot 2^{\mathrm{qd}_2(F)}\}$.*

Proof. If F has no free variables then $E_\Sigma^{graph(m)}(e) = \{(e,i) \mid 1 \leq i \leq m(e)\}$. Using Proposition 5, we can thus reduce the number of copies for every edge to the number $\mathrm{qd}_1(F) \cdot 2^{\mathrm{qd}_2(F)}$, without changing the truth value of F. □

The following corollary shows that every graph-statement of full monadic second-order logic can be encoded into a multiset formula.

Corollary 2. *Let G be a fixed template graph. A closed graph formula F in $\mathcal{L}2$ can be encoded into a logical formula $M_2(F)$ on multisets as follows. For any multiset $k \in E_G^\oplus$, let C_k be the conjunction over the following formulae:*

- *$\#e = k(e)$ for every $e \in E_G$ satisfying $k(e) < \mathrm{qd}_1(F) \cdot 2^{\mathrm{qd}_2(F)}$ and*
- *$\#e \geq k(e)$ for every $e \in E_G$ satisfying $k(e) = \mathrm{qd}_1(F) \cdot 2^{\mathrm{qd}_2(F)}$.*

Define $M_2(F)$ to be the disjunction of all C_k such that $k \in E_G^\oplus$, $graph(k) \models F$ and $k(e) \leq \mathrm{qd}_1(F) \cdot 2^{\mathrm{qd}_2(F)}$ for every $e \in E_G$.

Then $graph(m) \models F \iff m \models M_2(F)$ for every $m \in E_G^\oplus$.

Proof. Let $m \in E_G^\oplus$ be an arbitrary multiset and let m' be a multiset defined as in Corollary 1, i.e. $m'(e) = \min\{m(e), \mathrm{qd}_1(F) \cdot 2^{\mathrm{qd}_2(F)}\}$. for $e \in E_G$.

If $graph(m) \models F$ then, by Corollary 1, $graph(m') \models F$. Hence, by definition of M_2, $C_{m'}$ appears as a disjunct in $M_2(F)$. Since, clearly, $m \models C_{m'}$, we conclude that $m \models M_2(F)$.

Vice versa, let $m \models M_2(F)$. Then $m \models C_k$ for some $k \in E_G^\oplus$ and $graph(k) \models F$. By the shape of C_k, it is immediate to see that this implies $k = m'$. Therefore $graph(m') \models F$, and thus, by Corollary 1, $graph(m) \models F$. □

To conclude let us show how the general schema outlined at the end of Section 2 applies to our running example. We want to verify that Sys satisfies a safety property, i.e., the absence of vicious cycles, including two distinct P_2 processes, in all reachable graphs. Let $\Box L_\mu$ be a fragment of the μ-calculus without negation and "possibility operator" $\Diamond$ (see [10]), where basic predicates are formulae F taken from our graph logic $\mathcal{L}2$, which can be typed as "reflected by graph morphisms", i.e., such that $F :\leftarrow$ is provable. The property of interest can be expressed in $\Box L_\mu$ as:

$$T_{NC} = \mu\varphi.(NC_{P_2} \land \Box\varphi\ )$$

where NC_ℓ is the formula considered in a previous example. Then T_{NC} can be translated into a formula over markings, by translating its graph formula components according to the techniques described in Sections 6 and 7. This will lead to the formula $M_2(T_{NC}) = \mu\varphi.(M_2(NC_{P_2}) \wedge \Box\varphi\ )$. By the results in this paper and by the results in [2], for T in $\Box L_\mu$, if $\mathcal{U}(\mathsf{Sys}) \models M_2(T)$ then $\mathsf{Sys} \models T$. Therefore the formula T_{NC} can be checked by verifying $M_2(T_{NC})$ on the Petri net component of the approximated unfolding. In this case it can be easily verified that $M_2(T_{NC})$ actually holds in $\mathcal{U}(\mathsf{Sys})$ and thus we conclude that Sys satisfies the desired property.

8 Conclusion

We have presented a logic for specifying graph properties, useful for the verification of graph transformation systems. A type system allows us to identify formulae of this logic reflected by edge-bijective morphisms, which can therefore be verified on the covering, i.e., on the finite Petri graph approximation of a GTS. Furthermore we have shown how, given a fixed approximation of the original system, we can perform quantifier-elimination and encode these formulae into boolean combination of atomic predicates on multisets. Combined with the approximated unfolding algorithm of [1], this gives a method for the verification and analysis of graph transformation systems. This form of abstraction is different from the usual forms of abstract interpretation since it abstracts the *structure* of a system rather than its *data*. Maybe the closest relation is shape analysis, abstracting the data structures of a program [11, 15].

We would like to add some remarks concerning the practicability of this approach: we are currently developing an implementation of the approximated unfolding algorithm, which inputs and outputs graphs in the Graph Exchange Language (GXL) format, based on XML. It remains to be seen up to which size of a GTS the computation of the approximation is still feasible.

Furthermore encoding a formula into multiset logic may result in a blowup of the size of the formula which is at least exponential. However, provided that formulae are rather small if compared to the size of the system or its approximation, this blowup should be manageable. It is also conceivable to simplify a formula during its encoding (see the example at the end of Section 6). The encoding itself is not yet implemented, but we plan to do so in the future.

Finally the Petri net produced by the approximated unfolding algorithm and the formula itself have to be analysed by a model checker or a similar tool, based on the procedures described in [8, 7, 9]. Note that formulae on multisets can not be combined with the temporal operators of CTL* in an arbitrary way. First, we have to make sure that the resulting formula is still reflected, with respect to the simulation, hence no existential path quantification is allowed. Furthermore, arbitrary combinations of the temporal operators "eventually" and "generally" might make the model-checking problem undecidable. However, important fragments are still decidable, for example a property like "all reachable graphs satisfy F", where F is a multiset formula, can be checked. As far as we know, there is not much tool support for model-checking unbounded Petri nets, but these algorithms usually rely on the computation of the coverability graph of a Petri net, which is a well-studied problem [13].

Currently we are mainly interested in proving safety properties, liveness properties require some more care (see [12]). Another interesting line of future research is to adopt techniques used for the analysis of transition systems specified by integer constraints [5].

Acknowledgements: We are very grateful to Andrea Corradini for his contribution to the development of the approximated unfolding technique on which this paper is based. We would also like to thank Ingo Walther who is currently working on an implementation. We are also grateful to the anonymous referees for their valuable comments.

References

1. Paolo Baldan, Andrea Corradini, and Barbara König. A static analysis technique for graph transformation systems. In *Proc. of CONCUR '01*, pages 381–395. Springer-Verlag, 2001. LNCS 2154.
2. Paolo Baldan and Barbara König. Approximating the behaviour of graph transformation systems. In *Proc. of ICGT '02 (International Conference on Graph Transformation)*, pages 14–29. Springer-Verlag, 2002. LNCS 2505.
3. Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 1999.
4. B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1: Foundations*, chapter 5. World Scientific, 1997.
5. Giorgio Delzanno. Automatic verification of parameterized cache coherence protocols. In *Proc. of CAV '00*, pages 53–68. Springer-Verlag, 2000. LNCS 1855.
6. Wilfrid Hodges. *Model Theory.* Cambridge University Press, 1993.
7. R. Howell and L. Rosier. Problems concerning fairness and temporal logic for conflict-free Petri net. *Theoretical Computer Science*, 64:305–329, 1989.
8. Rodney R. Howell, Louis E. Rosier, and Hsu-Chun Yen. A taxonomy of fairness and temporal logic problems for Petri nets. *Theoretical Computer Science*, 82:341–372, 1991.
9. Petr Jančar. Decidability of a temporal logic problem for Petri nets. *Theoretical Computer Science*, 74:71–93, 1990.
10. Claire Loiseaux, Susanne Graf, Joseph Sifakis, Ahmed Bouajjani, and Saddek Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–35, 1995.
11. Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis.* Springer-Verlag, 1999.
12. Amir Pnueli, Jessie Xu, and Lenore Zuck. Liveness with $(0, 1, \infty)$-counter abstraction. In *Proc. of CAV '02*, pages 107–122. Springer-Verlag, 2002. LNCS 2404.
13. W. Reisig. *Petri Nets: An Introduction.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1985.
14. Abraham Robinson. *Introduction to Model Theory and to the Metamathematics of Algebra.* North-Holland, 1963.
15. M. Sagiv, T. Reps, and R. Wilhelm. Solving shape-analysis problems in languages with destructive updating. In *Proc. of POPL '96*, pages 16–31. ACM Press, 1996.