Replace this file with `prentcsmacro.sty` for your meeting,
or with `entcsmacro.sty` for your meeting. Both can be
found at the ENTCS Macro Home Page.

Verifying a Behavioural Logic for Graph Transformation Systems [1]

Paolo Baldan [2]

Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy

and

Andrea Corradini [3]

Dipartimento di Informatica, Università di Pisa, Italy

and

Barbara König [4]

Institutsverbund Informatik, University of Stuttgart, Germany

and

Bernhard König [5]

Department of Mathematics, University of California, Irvine, USA

Abstract

We propose a framework for the verification of behavioural properties of systems modelled as graph transformation systems. The properties can be expressed in a temporal logic which is basically a μ-calculus where the state predicates are formulae of a monadic second order logic, describing graph properties. The verification technique relies on an algorithm for the construction of finite over-approximations of the unfolding of a graph transformation system.

[1] Partially supported by the Italian MIUR Project COFIN 2001013518 CoMeta, the EU FET – GC Project IST-2001-32747 Agile and the EC RTN 2-2001-00346 SegraVis.

[2] Email: `baldan@dsi.unive.it`
[3] Email: `andrea@di.unipi.it`
[4] Email: `koenigba@fmi.uni-stuttgart.de`
[5] Email: `bkoenig@math.uci.edu`

1 Introduction

Graph transformation systems (GTSs) are recognised as a powerful specification formalism for concurrent and distributed systems [12], generalising Petri nets. Their truly concurrent behaviour has been deeply studied and a consolidated theory of concurrency is now available. In particular, several semantics of Petri nets, like process and unfolding semantics, have been extended to GTSs (see, e.g., [8,26,1,2]). However, concerning automated verification, while several approaches exist for Petri nets, ranging from the calculus of invariants [25] to model checking based on finite complete prefixes [23], the rich literature on GTSs does not contain many contributions to the static analysis of such systems (see [20,21,14]). Interestingly, several formalisms for concurrency and mobility can be encoded as graph transformation systems, in a way that verification techniques for graph transformation systems potentially carry over such formalisms.

This paper describes a framework, developed in [5,6,4], where behavioural properties of a system described as a GTS can be specified and verified.

A Logic for Behavioural Properties of GTSs. The logic used here to specify behavioural properties of GTSs is the temporal logic $\mu\mathcal{L}2$, which can be seen as a variation of the μ-calculus. The formulae of $\mu\mathcal{L}2$ are generated by closing *state predicates* under temporal modalities ($\Box$ and $\Diamond$), fixed-point operators (μ and ν), and standard logical connectives. Negation is classical. In turn, state predicates, which are used to express the graph properties of interest, are formed according to the monadic second-order logic $\mathcal{L}2$ on graphs, where quantification is allowed over (sets of) edges (see, e.g., [9].) Interesting graph properties can be expressed in $\mathcal{L}2$, like the non-existence or non-adjacency of edges with specific labels, and the absence of certain paths or of certain cycles. Such properties may be used to represent in the graph transformation model relevant properties of the system at hand, like security properties or deadlock-freedom.

Approximating the Behaviour of GTSs. A basic ingredient for the verification of $\mu\mathcal{L}2$ is a technique, proposed in [5,6], for approximating the behaviour of GTSs by means of finite Petri net-like structures, in the spirit of abstract interpretation of reactive systems [22]. More precisely, an approximated unfolding construction maps any given GTS $\mathcal{G}$ to finite structures, called *coverings* of $\mathcal{G}$, which provide "effective" (over-)approximations of the behaviour of $\mathcal{G}$.

The accuracy of the approximation can be chosen by the user and arbitrarily increased. Essentially one can require the approximation to be exact up to a certain causal depth k, thus obtaining the so-called *k-covering* $\mathcal{C}^k(\mathcal{G})$ of $\mathcal{G}$. The coverings are *Petri graphs*, i.e., structures consisting of a Petri net with a graphical structure over places. Each $\mathcal{C}^k(\mathcal{G})$ over-approximates the behaviour of $\mathcal{G}$ in the sense that every computation of $\mathcal{G}$ is mapped to a valid computation of $\mathcal{C}^k(\mathcal{G})$ and every graph reachable from the start graph can be mapped homomorphically to (the graphical component of) $\mathcal{C}^k(\mathcal{G})$ and its image is reachable in the Petri graph. Therefore, given a property over graphs *reflected* by graph morphisms, if it holds for all states reachable in the abstraction $\mathcal{C}^k(\mathcal{G})$ then it also holds for all reachable graphs in $\mathcal{G}$.

Verifying Behavioural Properties of GTSs. Relying on the approximations of the unfolding, we propose a technique for reducing the verification of a $\mu\mathcal{L}2$ formula over a GTS $\mathcal{G}$ to the verification of a corresponding multiset formula over (the Petri net component of) a covering of $\mathcal{G}$. More specifically, fixed a covering $\mathcal{C}^k(\mathcal{G})$, we define a constructive translation of formulae in $\mu\mathcal{L}2$ into formulae over the Petri net underlying the abstraction $\mathcal{C}^k(\mathcal{G})$. This is done in two steps.

- First, any state predicate F in $\mathcal{L}2$ 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. The translation is a kind of quantifier elimination procedure which encodes 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). This somehow surprising fact can be understood by recalling that the graph underlying $\mathcal{C}^k(\mathcal{G})$ is finite and fixed after computing the abstraction.
- Then any temporal formula in $\mu\mathcal{L}2$ over $\mathcal{G}$ is translated to a temporal formula over the Petri graph by simply translating its $\mathcal{L}2$-subformulae as sketched above, and keeping the "temporal part" untouched.

Altogether, these results allow us to verify behavioural properties of a GTS, expressed in a suitable fragment of $\mu\mathcal{L}2$, by reusing existing model-checking techniques for Petri nets. In fact, consider a formula T in $\Box\mu\mathcal{L}2$ (i.e., not containing in the temporal part the modality $\Diamond$ nor negation) to be checked over a GTS $\mathcal{G}$. If the state predicates in T are reflected by graph morphisms, by the construction mentioned above we can translate T into a formula $\hat{T}$ over the Petri net underlying a covering of $\mathcal{C}^k(\mathcal{G})$ of $\mathcal{G}$. (The restriction to $\Box\mu\mathcal{L}2$ is necessary because $\mathcal{C}^k(\mathcal{G})$ *over-approximates* $\mathcal{G}$.) Then, by general results from abstract interpretation [22], $\hat{T}$ can be checked over the Petri net underlying $\mathcal{C}^k(\mathcal{G})$. A type inference system is introduced which characterises a subclass of formulae in the logic $\mathcal{L}2$ which are reflected by graph morphisms. Hence, the requirement over state predicates in T can be verified by checking that any such predicate can be typed as "reflected" in the mentioned type system.

We recall that 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 [17,16,18].

For the sake of simplicity, although the approximation method of [5,6] was designed for hypergraphs, in this paper we stick to directed graphs. Moreover, we mention that, although not discussed in this document, a dual theory involving under-approximations of the behaviour of GTSs has been developed (see [4]).

In the rest of the paper, after introducing the class of graph transformation systems handled by our approach, we will present the monadic second-order logic $\mathcal{L}2$ of graph formulae, and the temporal logic built on it, called $\mu\mathcal{L}2$. Next we shall summarise the approximation technique for GTSs developed in [5], briefly mentioning some results from [6]. Finally we will propose a method for verifying

$\mu\mathcal{L}2$ formulae over GTS's: this makes use of a 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), as well as of an encoding of these formulae into quantifier-free multiset formulae on the markings of Petri nets.

2 Graph Transformation Systems

In this section we introduce the class of graph transformation systems considered in this paper, which are basically graph rewriting systems in the double-pushout approach [11], with some restrictions.

We first define graphs and structure-preserving morphisms on graphs. We will assume that Λ denotes a fixed and finite set of labels.

Definition 2.1 (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, source and target functions $s_G, t_G : E_G \to V_G$ and a function $l_G : E_G \to \Lambda$ labelling the edges.

A *graph morphism* $\varphi : G_1 \to G_2$ is a pair $\langle \varphi_V : V_{G_1} \to V_{G_2}, \varphi_E : E_{G_1} \to E_{G_2} \rangle$ of mappings 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 graph transformation system and the corresponding rewriting relation.

Definition 2.2 (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 $\alpha : V_L \to V_R$ is an injective function.

A *match* of a rewriting rule r in a graph G is a morphism $\varphi : L \to G$ which is injective on edges. We can apply r to a match φ in G obtaining a new graph H, written $G \stackrel{r}{\Rightarrow} H$. 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. The resulting graph is extended by adding the new nodes in R (i.e., the nodes in $V_R - \alpha(V_L)$) and all the edges of R. Observe that the (images of the) nodes in L are preserved, i.e., they are not affected by the rewriting step.

Example 2.3 Consider a variant of the Dining Philosopher system where processes compete for resources R_1 and R_2, a process needs both resources in order to perform some task, but processes are not organised in a cyclic structure and they can

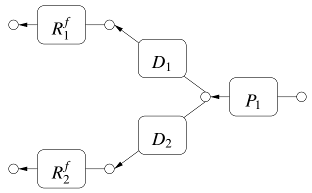

Fig. 1. Start graph of RPh with a process and two free resources.

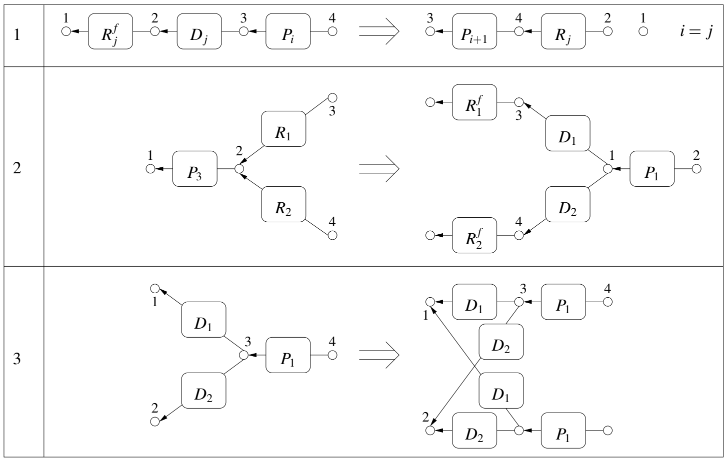

Fig. 2. Rewriting rules of the GTS RPh.

reproduce themselves. The system is represented as a GTS RPh 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 the first resource, a process waiting for the second resource, 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 that process. The initial scenario for RPh is represented in Fig. 1, with a single process P_1 asking for both resources.

The rewriting rules of RPh 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 it becomes 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 competing for the same resources. The natural numbers $1, 2, 3, \ldots$ which decorate nodes in the left- and right-hand side of rules implicitly represent the mapping α.

With the given rules, deadlocks are avoided by forcing each process to acquire the resources in a fixed ordering: first R_1 and then R_2. 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).

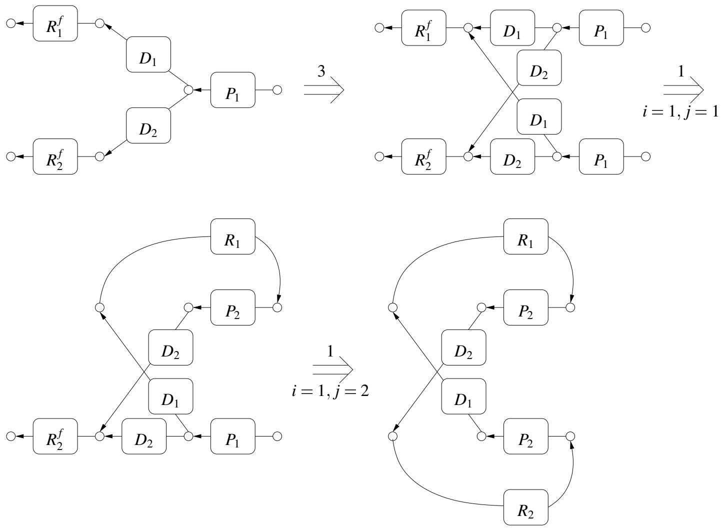

Fig. 3. A vicious cycle representing a deadlock.

3 A Logic for Graph Transformation Systems

This section presents a behavioural logic for GTSs. It is essentially a variant of the propositional μ-calculus (i.e., a temporal logic enriched with fixed-point operators) where propositional symbols range over arbitrary *state predicates*, characterising static graph properties, which are expressed in a monadic second-order logic.

3.1 A Monadic Second-Order Logic for Graphs

We first introduce the monadic second-order logic $\mathcal{L}2$ for specifying graph properties, i.e., "static" properties of system states. Quantification is allowed over edges, but not over nodes (as, e.g., in [9]).

Definition 3.1 (Graph formulae) Let $\mathcal{X}_1 = \{x, y, z, \dots\}$ be a set of (first-order) edge variables and $\mathcal{X}_2 = \{X, Y, Z, \dots\}$ be a set of (second-order) variables ranging over edge sets. The set of *graph formulae* of logic $\mathcal{L}2$ is defined as follows, where $\ell \in \Lambda$

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

We denote by $\mathit{free}(F)$ and $\mathit{Free}(F)$ the sets of first-order and second-order variables, respectively, occurring free in F.

The notion of satisfaction is defined in a straightforward way.

Definition 3.2 (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:

$$
\begin{aligned}
G \models_{\sigma,\Sigma} x = y &\iff \sigma(x) = \sigma(y)\\
G \models_{\sigma,\Sigma} s(x) = s(y) &\iff s_G(\sigma(x)) = s_G(\sigma(y))\\
G \models_{\sigma,\Sigma} lab(x) = \ell &\iff l_G(\sigma(x)) = \ell\\
G \models_{\sigma,\Sigma} x \in X &\iff \sigma(x) \in \Sigma(X)\\
G \models_{\sigma,\Sigma} \forall X.F &\iff G \models_{\sigma,\Sigma'} F \text{ for any } \Sigma' \text{ such that } \Sigma'(Y) = \Sigma(Y)\\
& \text{for } Y \in \mathcal{X}_2 - \{X\}, \text{ and } \Sigma'(X) \in \mathcal{P}(E_G),
\end{aligned}
$$

Example 3.3 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 RPh. It is based on the formula $NP(x,y)$, which says that there is no path connecting the edges x and y, stating that a set that contains at least all edges reachable from x does not contain y necessarily.

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

A standard compactness argument shows that NC_ℓ cannot be stated in first-order logic, a fact which motivates our choice of considering a second-order logic.

3.2 Introducing a Temporal Dimension

The behavioural logic for GTSs, called $\mu\mathcal{L}2$, is a variant of the propositional μ-calculus where propositional symbols range over formulae from $\mathcal{L}2$.

Definition 3.4 (Logic over GTSs) The syntax of $\mu\mathcal{L}2$ formulae is the following:

$$f ::= A \mid X \mid \Diamond f \mid \Box f \mid \neg f \mid f_1 \vee f_2 \mid f_1 \wedge f_2 \mid \mu X.f \mid \nu X.f$$

where A ranges over closed formulae in $\mathcal{L}2$ and $X \in \mathcal{X}$ are *proposition variables*.

The formulae are evaluated over a *graph transition system* $T = (Q, \rightarrow)$, i.e., a transition system where the set of states Q consists of (isomorphism classes of) graphs. This can be thought of as the abstract representation of the behaviour of a graph grammar. Intuitively, an atomic proposition A holds in any state q satisfying A according to Definition 3.2. A formula $\Diamond f$ / $\Box f$ holds in a state q if some / any single step leads to a state where f holds. Note that (as in [22]) the operators $\Box$ and $\Diamond$ only refer to the next step and not (as defined elsewhere) to the whole computation. The connectives $\neg, \vee, \wedge$ are interpreted in the usual way. The formulae $\mu X.f$ and $\nu X.f$ represent the *least* and *greatest fixed point* over X, respectively. When a transition system T has a distinguished *initial state* q_0, we say that T *satisfies a (closed) formula* f, written $T \models f$, if the initial state q_0 of T satisfies f. Since the logic is classical, $\Diamond$ and ν could be defined in terms of $\Box$ and μ. All the operators are inserted explicitly since later we will restrict to negation-free fragments of $\mu\mathcal{L}2$.

The fragment of $\mu\mathcal{L}2$ without negation and box operator is denoted by $\Diamond\mu\mathcal{L}2$. By dropping negation and the diamond operator we obtain the fragment $\Box\mu\mathcal{L}2$. Some typical *liveness* properties of the form "eventually A" (i.e., $\mu X.(A \vee \Diamond X)$) can be expressed in the fragment $\Diamond\mu\mathcal{L}2$, whereas some typical *safety* properties of the form "always A" (i.e., $\nu X.(A \wedge \Box X)$) can be expressed in the fragment $\Box\mu\mathcal{L}2$.

Example 3.5 Consider the system RPh in our running example. We would like to express the fact that, according to the design intentions, RPh 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.

Notice that this is a safety property which can be encoded in $\Box\mu\mathcal{L}2$ as follows:

$$T_{NC} = \nu\varphi.(NC_{P_2} \wedge \Box\varphi\ )$$

where NC_ℓ is the formula considered in a previous example.

4 Approximated Unfolding Construction

In this section we sketch the algorithm, introduced in [5,6], for the construction of finite over-approximations of the unfolding of a graph transformation system. This plays a crucial role in the verification process of the logic $\mu\mathcal{L}2$.

4.1 Petri Graphs

In order to approximate graph transformation systems we will use Petri net-like structures, called Petri graphs, originally introduced in [5].

To deal with Petri nets we first need to recall some basic notation concerning multisets. Given a set A we will denote by $A^\oplus$ the free commutative monoid over A, whose elements will be called *multisets* over A. We will sometimes identify $A^\oplus$ with the set of functions $m{:}A \to \mathbb{N}$ 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 $f{:}A \to B$, by $f^\oplus{:}A^\oplus \to B^\oplus$ we denote its monoidal extension, i.e., $f^\oplus(m)(b) = \sum_{f(a)=b} m(a)$ for every $b \in B$.

Petri graphs over a given graph transformation system $\mathcal{G}$ are basically Petri nets equipped with an additional graphical structure where the places play the role of edges, while the transitions represent applications of the rules of $\mathcal{G}$.

Definition 4.1 (Petri graph) 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 and

- $N = (E_G, T_N, {}^\bullet(), ()^\bullet, p_N)$ 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, mapping each transition to a corresponding rule;
- $m_0 \in (E_G)^\oplus$ 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 $\mathcal{G}$).

We shall write $m\,[r\rangle\, m'$ if a transition labelled by r is enabled at marking m and its firing produces m'. 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.

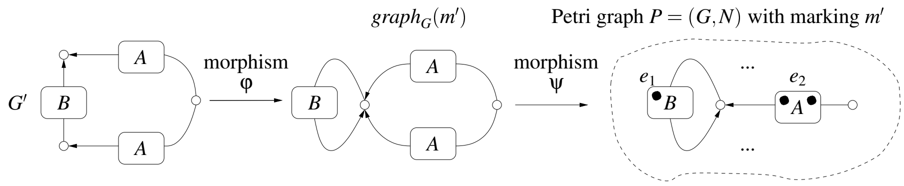

Fig. 4. A pair (G', m') contained in a simulation.

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

Definition 4.2 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 $graph_G(m)$, is the graph H without isolated nodes (unique up to isomorphism) such that there exists a morphism $\psi: H \to G$ injective on nodes with $\psi^{\oplus}(E_H) = m$. More explicitly, the graph H is 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 $l_H((e,i)) = l_G(e)$.

An example of Petri net marking and its generated graph can be found in Fig. 4.

4.2 Approximated Unfolding Algorithm

Given a GTS $(G_0, \mathcal{R})$, with some minor constraints on the format of rewriting rules (see [5,6]), for any $k \in \mathbb{N}$ we can construct a Petri graph approximation of $(G_0, \mathcal{R})$, called *(k-)covering* and denoted by $\mathcal{C}^k(G_0, \mathcal{R})$. Intuitively, the k-covering is exact up to causal depth k, i.e., any computation consisting of less than k (possibly concurrent) steps, is represented in $\mathcal{C}^k(G_0, \mathcal{R})$ without any loss of information.

In order to make this more formal, given a Petri graph $P = (N, G)$, let the *causal relation* $<$ be the least transitive relation over transitions such that, if $t_1{}^\bullet \cap {}^\bullet t_2 \neq \emptyset$ then $t_1 < t_2$. Then define the *depth of a transition* t to be the length of the longest sequence $t_0 < t_1 < \ldots < t_n < t$. The depth of an edge is the maximum among the depths of transitions which contain the edge in their post-set.

Then the covering $\mathcal{C}^k(G_0, \mathcal{R})$ 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.

(i) $P_0 = (G_0, N_0, m_0)$, where the net N_0 contains no transitions and $m_0 = E_{G_0}$.

(ii) 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 $\varphi: L \to G_i$ 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 $\varphi, \varphi' : L \to G_i$, at depth greater than or equal to k, such that
- $\varphi^{\oplus}(E_L)$ and $\varphi'^{\oplus}(E_L)$ are coverable in N_i and
- the second match causally depends on the transition unfolding the first one.

Then merge the two matches by setting $\varphi(e) \equiv \varphi'(e)$ for each $e \in E_L$ and

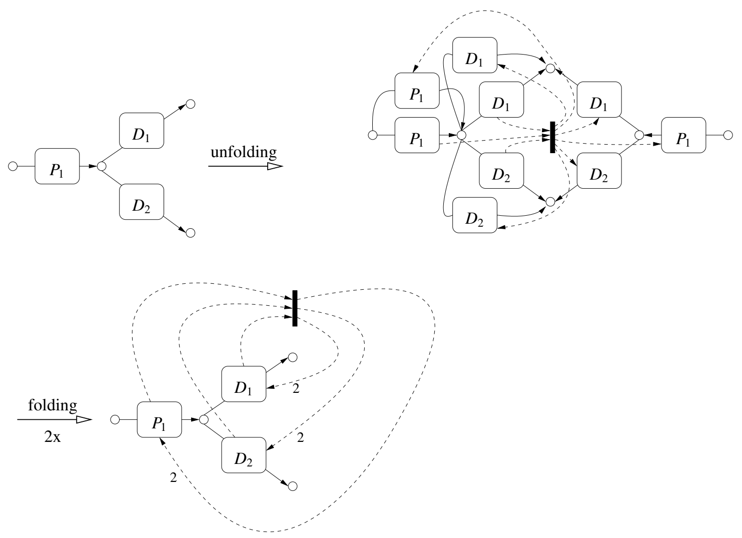

Fig. 5. An unfolding and two folding steps.

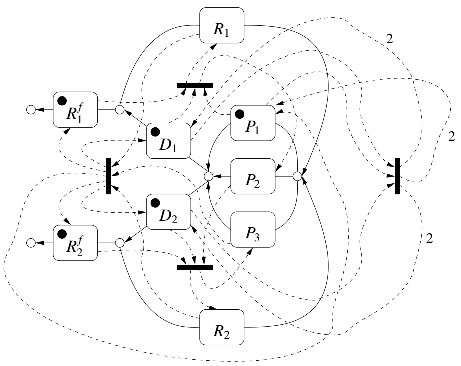

Fig. 6. The Petri graph $\mathcal{C}^0(\mathsf{RPh})$ computed as covering of RPh.

factoring through the resulting equivalence relation $\equiv$.

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 n 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 $\mathcal{C}^0(\mathsf{RPh})$ in Fig. 6, where the initial marking is represented by tokens.

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

Proposition 4.3 (Abstraction) *Let $\mathcal{G} = (G_0,\mathcal{R})$ be a graph transformation system and let $\mathcal{C}^k(\mathcal{G}) = (G,N,m_0)$ be its k-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 $\mathcal{C}^k(\mathcal{G})$. 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* $G' \stackrel{r}{\Rightarrow} G''$, *then there exists a marking* m'' *with* $m\,[r\rangle\, m''$ *and* $(G'', m'') \in S$;
- *for every* $(G', m') \in S$ *there exists an edge-bijective graph morphism* $\varphi\colon G' \to$ $\mathit{graph}_G(m')$.

The above result will allow us to use existing results concerning abstractions of reactive systems [7,22]. Consider the property T_{NC} in Example 3.5 expressing the absence of dead-locks in system RPh. Observe that the graph property NC_{P_2}, i.e., the absence of vicious cycles where P_2-labelled edges (processes holding one resource and waiting for a second one) occur twice is reflected by edge-bijective graph morphisms. Hence, by using Proposition 4.3, if we can prove it on the covering $\mathcal{C}^0(\mathsf{RPh})$ of Fig. 6, we could deduce that it holds for the original system RPh as well. Observe that actually, in this case, even the stronger property "$\#e \leq 1$", i.e., "e contains at most one token", 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.

5 Verifying $\mu\mathcal{L}2$ Formulae on Graph Transformation Systems

In this section we show how the verification of properties of a GTS $\mathcal{G}$, expressed in the logic $\mu\mathcal{L}2$, can be reduced to the verification of suitable properties of the Petri net underlying a covering of $\mathcal{G}$. This is done by viewing our technique as a specific instance of abstract interpretation [10,19], and exploiting some results from [22].

The idea is the following. Let $\mathcal{G}$ be a GTS and let $\mathcal{C}^k(\mathcal{G})$ be its k-covering. By Proposition 4.3, $\mathcal{C}^k(\mathcal{G}) = (G, N, m_0)$ "approximates" $\mathcal{G}$ via a simulation consisting of pairs (G', m') such that G' can be mapped to $\mathit{graph}_G(m')$ (see, e.g., Fig. 4) via an edge-bijective morphism. Given a formula on graphs F in $\mathcal{L}2$, expressing a state property in $\mathcal{G}$, a corresponding formula $M(F)$ on the markings of $\mathcal{C}^k(\mathcal{G})$ is constructed such that, for any pair in the simulation,

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

and thus, in a sense, F can be safely checked over the approximation $\mathcal{C}^k(\mathcal{G})$. This will be obtained in two steps. First, we will define a type system which allows us to identify formulae F which are reflected by edge-bijective morphisms, ensuring that $\mathit{graph}_G(m') \models F$ implies $G' \models F$. Then, we will encode F into a propositional formula $M(F)$ on multisets such that $m' \models M(F) \iff \mathit{graph}_G(m') \models F$.

Finally, consider the temporal logic $\mu\mathcal{L}2$ over GTSs. For suitable fragments of such logics, e.g., the fragment $\Box\mu\mathcal{L}2$ without negation and the "possibility operator" $\Diamond$, by Proposition 4.3 and exploiting general results in [22], a temporal formula T over $\mathcal{G}$, where state predicates can be typed as "reflected", can be translated to a formula $M(T)$ over markings (translating the state predicates as described above), such that, if $\mathcal{C}^k(\mathcal{G}) \models M(T)$ then $\mathcal{G} \models T$, i.e., T is valid for the original GTS.

$$s(x) = s(y),\, s(x) = t(y),\, t(x) = t(y)\!:\rightarrow \qquad x = y,\, lab(x) = \ell,\, x \in X\!:\leftrightarrow$$

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

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

Fig. 7. A type system for preservation and reflection of $\mathcal{L}2$ formulae by edge-bijective morphisms.

5.1 Preservation and Reflection of Graph Formulae

Preservation and reflection of graph formulae by graph morphisms are defined as follows.

Definition 5.1 Let F be a formula in $\mathcal{L}2$ and $\varphi : G_1 \to G_2$ be a graph morphism. We say that F is *preserved (resp. reflected) by* φ if for all valuations $\sigma : \mathit{free}(F) \to E_{G_1}$ and $\Sigma : \mathit{Free}(F) \to \mathcal{P}(E_{G_1})$, $G_1 \models_{\sigma,\Sigma} F$ implies (resp. is implied by) $G_2 \models_{\varphi\circ\sigma,\varphi\circ\Sigma} F$.

We are interested in syntactic criteria characterising classes of graph formulae reflected, respectively preserved, by all edge-bijective graph morphisms. Similar considerations on reflection and preservation on morphisms can be found in [15]. Here we provide a technique which works for general second-order monadic formulae. It is based on the type system of Fig. 7 which allows to prove judgements of the form $F :\rightarrow$ (resp. $F :\leftarrow$), meaning that the $\mathcal{L}2$-formula F is preserved (resp. reflected) by all edge-bijective graph morphisms. In the typing rules, it is intended that $\rightarrow^{-1} = \leftarrow$ and $\leftarrow^{-1} = \rightarrow$. Moreover $F :\leftrightarrow$ is a shorthand for $F :\rightarrow$ and $F :\leftarrow$, while $F_1, F_2 : d$ stands for $F_1 : d$ and $F_2 : d$.

Example 5.2 The judgements $\mathit{NP}(x,y)\!:\leftarrow$ and $\mathit{NC}_\ell\!:\leftarrow$ are provable using the type system of Fig. 7, thus the absence of paths and of vicious cycles is reflected by edge-bijective morphisms.

The proposed type system can be shown easily to be correct, but it is not complete. In fact, it is possible to show that the set of closed first-order formulae which are preserved (resp. reflected) by *edge-bijective* morphisms is undecidable. Therefore, *a fortiori*, not all $\mathcal{L}2$-formulae which are preserved or reflected are captured by the above type system.

5.2 A Propositional Logic on Multisets

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

Definition 5.3 (Multiset formulae) 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 $\#a = c$ with the obvious meaning. E.g., $(\#a \geq c)$ stands for $\neg(\#a \leq c-1)$ if $c > 0$ and for *true* otherwise.

5.3 Encoding Graph Logic into Multiset Logic

We show how graph formulae can be encoded into "equivalent" multiset formulae. More precisely, for a fixed Petri graph $P = (G, N, m_0)$ the aim is to find an encoding M_1 of $\mathcal{L}2$-formulae into multiset formulae such that $graph_G(m) \models F \iff m \models M_1(F)$ for every marking m of P and every closed graph formula F.

We actually propose two encodings: an inductive one, which works only for first-order formulae, and a general one which works for the whole set of graph formulae.

Inductive Encoding. The encoding M_1 of first-order graph formulae is based on the following observation. Every graph $graph_G(m)$ 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 $graph_G(m)$ has n copies $e_1, \ldots, e_n$ 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 Q on the free variables $free(F)$, telling us which variables are mapped to the same edge, and a mapping ρ from $free(F)$ 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 Q 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 possibile choices for x: either x is instantiated with the same edge as another free variable y, and in this case x and y should be in the same set of the partition Q. Or x is instantiated with a new copy of an edge in G. In this case, a new set $\{x\}$ is added to Q 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 $Q \subseteq \mathcal{P}(A)$ of A, where every element of Q represents an equivalence class. We will write $x\,Q\,y$ whenever x, y are in the same equivalence class. Furthermore we assume that each equivalence Q is associated with a function $rep : Q \to A$ 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 aQa' and a fixed $b \in B$ we define $n_{Q,f}(b) = |\{k \in Q \mid f(rep(k)) = b\}|$, i.e., $n_{Q,f}(b)$ is the number of sets in the partition Q that are mapped to b.

Definition 5.4 (Encoding of first-order graph formulae) Let G be a finite directed graph, let F be a graph formula in the first-order fragment of $\mathcal{L}2$, let $\rho : \mathit{free}(F) \to E_G$ and let $Q \subseteq \mathcal{P}(\mathit{free}(F))$ be an equivalence relation such that $x\,Q\,y$ implies $\rho(x) = \rho(y)$ for all $x, y \in \mathit{free}(F)$. The *encoding* M_1 is defined as follows:

$$
\begin{aligned}
M_1[\neg F, \rho, Q] &= \neg M_1[F, \rho, Q] \\
M_1[F_1 \vee F_2, \rho, Q] &= M_1[F_1, \rho, Q] \vee M_1[F_1, \rho, Q] \\
M_1[F_1 \wedge F_2, \rho, Q] &= M_1[F_1, \rho, Q] \wedge M_1[F_1, \rho, Q] \\
M_1[x = y, \rho, Q] &= \begin{cases} \mathit{true} & \text{if } x\,Q\,y \\ \mathit{false} & \text{otherwise} \end{cases} \\
M_1[\mathit{lab}(x) = \ell, \rho, Q] &= \begin{cases} \mathit{true} & \text{if } l_G(\rho(x)) = \ell \\ \mathit{false} & \text{otherwise} \end{cases} \\
M_1[s(x) = s(y), \rho, Q] &= \begin{cases} \mathit{true} & \text{if } s_G(\rho(x)) = s_G(\rho(y)) \\ \mathit{false} & \text{otherwise} \end{cases} \\
& \text{the formulae } t(x) = t(y) \text{ and } s(x) = t(y) \text{ are treated analogously} \\
M_1[\exists x.F, \rho, Q] &= \bigvee_{k \in Q} (M_1[F, \rho \cup \{x \mapsto \rho(\mathit{rep}(k))\}, Q \backslash \{k\} \cup \{k \cup \{x\}\}]) \vee \\
& \quad \bigvee_{e \in E_G} (M_1[F, \rho \cup \{x \mapsto e\}, Q \cup \{\{x\}\}] \wedge (\#e \geq n_{Q,\rho}(e) + 1)) \\
M_1[\forall x.F, \rho, Q] &= \bigwedge_{k \in Q} (M_1[F, \rho \cup \{x \mapsto \rho(\mathit{rep}(k))\}, Q \backslash \{k\} \cup \{k \cup \{x\}\}]) \wedge \\
& \quad \bigwedge_{e \in E_G} ((\#e \geq n_{Q,\rho}(e) + 1) \Rightarrow M_1[F, \rho \cup \{x \mapsto e\}, Q \cup \{\{x\}\}])
\end{aligned}
$$

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

General Encoding. General monadic second-order graph formulae in $\mathcal{L}2$ can be encoded into multiset formulae, but differently from the first-order case, the encoding is not defined inductively, even if quantifier elimination is still possible.

Proposition 5.5 *Let G be a fixed finite 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 $k(e) = \mathrm{qd}_1(F) \cdot 2^{\mathrm{qd}_2(F)}$.*

where $\mathrm{qd}_1(F)$ and $\mathrm{qd}_2(F)$ denote the first order and second order quantifier depth of F, defined in the obvious way. Define $M_2(F)$ to be the disjunction of all C_k such that $k \in E_G^{\oplus}$, $\mathit{graph}_G(k) \models F$ and $k(e) \leq \mathrm{qd}_1(F) \cdot 2^{\mathrm{qd}_2(F)}$ for every $e \in E_G$.

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

Intuitively, for every first-order quantifier we have to try every edge of the template graph. Furthermore we have to try every possible membership to the sets assigned to second-order variables, of which there are at most $\mathrm{qd}_2(F)$, hence there are at most $2^{\mathrm{qd}_2(F)}$ possible membership combinations.

Although a formal complexity analysis is beyond the scope of this paper, we mention that the inductive encoding offers the advantage of generally producing smaller propositional formulae, easier to simplify.

5.4 Verification of Temporal Formulae of $\mu\mathcal{L}2$

We need to recall some concepts from [22], the more basic one being the formalisation of abstraction given in terms of Galois connections.

Definition 5.6 (Galois connection) Let Q_1 and Q_2 be two sets of states. A *Galois connection* from $\mathcal{P}(Q_1)$ to $\mathcal{P}(Q_2)$ is a pair of monotonic functions (α,γ), with $\alpha : \mathcal{P}(Q_1) \to \mathcal{P}(Q_2)$ *(abstraction)* and $\gamma : \mathcal{P}(Q_2) \to \mathcal{P}(Q_1)$ *(concretisation)*, such that $id_{\mathcal{P}(Q_1)} \subseteq \gamma \circ \alpha$ and $\alpha \circ \gamma \subseteq id_{\mathcal{P}(Q_2)}$.

Galois connections can be defined on general partial orders, but in this context we only need to consider them over powerset lattices. Next we introduce $\langle\alpha,\gamma\rangle$-simulations which turn out to coincide with simulations in the sense of Milner.

Definition 5.7 ($\langle\alpha,\gamma\rangle$-simulation) Let $T_i = (Q_i, \to_i)$ with $i \in \{1,2\}$ be transition systems, where Q_i is a set of *states* and $\to_i \subseteq Q_i \times Q_i$ is the *transition relation*. Let furthermore (α,γ) be a Galois connection from $\mathcal{P}(Q_1)$ to $\mathcal{P}(Q_2)$.

We say that T_2 $\langle\alpha,\gamma\rangle$*-simulates* T_1, written $T_1 \sqsubseteq_{\langle\alpha,\gamma\rangle} T_2$, if $\alpha \circ pre[\to_1] \circ \gamma \subseteq pre[\to_2]$, where the function $pre[\to_i] : \mathcal{P}(Q_i) \to \mathcal{P}(Q_i)$ is defined by $pre[\to_i](Q) = \{q \in Q_i \mid \exists q' \in Q : q \to_i q'\}$.

Let T_1, T_2 be transition systems and let $\varphi : T_1 \to T_2$ be a *transition system morphism*, i.e., a function $\varphi : Q_1 \to Q_2$ such that for any $q, q' \in Q_1$ if $q \to_1 q'$ then $\varphi(q) \to_2 \varphi(q')$ (in other words, φ is a special kind of simulation). Then the pair (φ, φ^{-1}) is a Galois connection and furthermore $T_1 \sqsubseteq_{\langle\varphi,\varphi^{-1}\rangle} T_2$.

These definitions allow us to interpret the approximations of the behaviour of a graph grammar in this context. In fact, let us identify any graph grammar $\mathcal{G}$ with the obvious *graph transition system* generated by $\mathcal{G}$ (as described after Definition 3.4). Observe that also a Petri graph (G, N, m_0) over $\mathcal{G}$ can be seen as a graph transition system, by simply identifying any marking m with the graph $graph_G(m)$. In this view, it can be shown that for any $k \in \mathbb{N}$, for suitable α and γ

$$\mathcal{G} \sqsubseteq_{\langle\alpha,\gamma\rangle} \mathcal{C}^k(\mathcal{G}).$$

By exploiting the results of [22] regarding the preservation and reflection of modal μ-calculus formulae on transitions systems and the results in this paper we have the following.

Proposition 5.8 *Let $\mathcal{G} = (G_0, \mathcal{R})$ be a GTS, and let T be a $\mu\mathcal{L}2$-formula. Consider the formula $M_2(T)$ obtained by replacing any state predicate A in T by the multiset formula $M_2(A)$ as defined in Proposition 5.5. Then if $T \in \Box\mu\mathcal{L}2$ and for any state* *predicate A in T the judgement $A :\leftarrow$ is provable using the type system of Fig. 7, then for any $k \in \mathbb{N}$*

$$\mathcal{C}^k(\mathcal{G}) \models M_2(T) \quad \Rightarrow \quad \mathcal{G} \models T$$

The above result shows how to reduce the analysis of the full transition system of a graph grammar to the analysis of simpler transition systems, generated by Petri nets (underlying Petri graphs). These transition systems might still have infinitely many states, but there are several decidability results for fragments of the modal μ-calculus and other forms of temporal logics [13,17,18]. Analogous result can be proved by restricting state predicates to the first-oder fragment of $\mathcal{L}2$ and using the encoding M_1.

For instance, consider the safety property T_{NC} over system RPh of Example 3.5, i.e., the absence of vicious cycles including two distinct P_2 processes in all reachable graphs. The formula T_{NC} can be translated into a formula over markings, by translating its graph formula components according to the techniques described in Sections 5.3. This will lead to the formula $M_2(T_{NC}) = \nu\varphi.(M_2(NC_{P_2}) \wedge \Box\varphi)$. Since T_{NC} belongs to the $\Box\mu\mathcal{L}2$ fragment and the basic predicate NC_{P_2} can be typed as "reflected", i.e., the judgement $NC_{P_2} :\leftarrow$ is provable in the type system of Fig. 7, by Proposition 5.8, if $\mathcal{C}^k(\mathsf{RPh}) \models M_2(T_{NC})$ then $\mathsf{RPh} \models T_{NC}$. 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 $\mathcal{C}^0(\mathsf{RPh})$ and thus we conclude that RPh satisfies the desired property.

6 Conclusions

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. Moreover, fixed an 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 [5], 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 [24,27].

Some natural generalisations of the approach presented in this paper are under development. Firstly, we intend to extend the approach to general *hypergraph* rewriting, also considering the fact that the approximation method of [5,6] was indeed designed for hypergraphs. The extension to general hypergraphs will require some basic changes to the graph logic $\mathcal{L}2$. Furthermore, we intend to consider wider classes of GTSs where it is possible to specify that an hyperedge is *preserved* by a rule: this will require to consider Petri graphs having an underlying *contextual* Petri net [3].

References

[1] P. Baldan, A. Corradini, and U. Montanari. Concatenable graph processes: relating processes and derivation traces. In *Proceedings of ICALP'98*, volume 1443 of *LNCS*, pages 283–295. Springer Verlag, 1998.

[2] P. Baldan, A. Corradini, and U. Montanari. Unfolding and Event Structure Semantics for Graph Grammars. In W. Thomas, editor, *Proceedings of FoSSaCS '99*, volume 1578 of *LNCS*, pages 73–89. Springer Verlag, 1999.

[3] P. Baldan, A. Corradini, and U. Montanari. Contextual petri nets, asymmetric event structures and processes. *Information and Computation*, 171(1):1–49, 2001.

[4] P. Baldan, B. König, and B. König. A logic for analyzing abstractions of graph transformation systems. In R. Cousot, editor, *Proceedings of SAS'03*, volume 2694 of *LNCS*, pages 255–272. Springer Verlag, 2003.

[5] 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.

[6] 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.

[7] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 1999.

[8] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.

[9] 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.

[10] P. Cousot. Abstract interpretation. *ACM Computing Surveys*, 28(2), 1996.

[11] H. Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *Proceedings of the 1st International Workshop on Graph-Grammars and Their Application to Computer Science and Biology*, volume 73 of *LNCS*, pages 1–69. Springer Verlag, 1979.

[12] H. Ehrig, J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism and Distribution*. World Scientific, 1999.

[13] J. Esparza. Decidability of model-checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[14] R. Heckel. Compositional verification of reactive systems specified by graph transformation. In E. Astesiano, editor, *Proceedings of FASE'98*, volume 1382 of *LNCS*, pages 138–153. Springer Verlag, 1998.

[15] Wilfrid Hodges. *Model Theory*. Cambridge University Press, 1993.

[16] R. Howell and L. Rosier. Problems concerning fairness and temporal logic for conflict-free Petri net. *Theoretical Computer Science*, 64:305–329, 1989.

[17] 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.

[18] Petr Jančar. Decidability of a temporal logic problem for Petri nets. *Theoretical Computer Science*, 74:71–93, 1990.

[19] N.D. Jones and F. Nielson. Abstract interpretation: a semantics-based tool for program analysis. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Vol. 4: Semantic Modelling*, pages 527–636. Oxford University Press, 1995.

[20] M. Koch. *Integration of Graph Transformation and Temporal Logic for the Specification of Distributed Systems*. PhD thesis, Technische Universität Berlin, 2000.

[21] B. König. A general framework for types in graph rewriting. In *Proc. of FST TCS 2000*, volume 1974 of *LNCS*, pages 373–384. Springer Verlag, 2000.

[22] 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.

[23] K.L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.

[24] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.

[25] W. Reisig. *Petri Nets: An Introduction*. EACTS Monographs on Theoretical Computer Science. Springer Verlag, 1985.

[26] L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars*. PhD thesis, Technische Universität Berlin, 1996.

[27] Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. *TOPLAS (ACM Transactions on Programming Languages and Systems)*, 24(3):217–298, 2002.