Towards the Verification of Attributed Graph Transformation Systems*

Barbara König and Vitali Kozioura

Abteilung für Informatik und Angewandte Kognitionswissenschaft
Universität Duisburg-Essen, Germany

Abstract. We describe an approach for the verification of attributed graph transformation systems (AGTS). AGTSs are graph transformation systems where graphs are labelled over an algebra. We base our verification procedure on so-called approximated unfoldings combined with counterexample-guided abstraction refinement. Both techniques were originally developed for non-attributed systems. With respect to refinement we focus especially on detecting whether the spurious counterexample is caused by structural over-approximation or by an abstraction of the attributes which is too coarse. The technique is implemented in the verification tool AUGUR 2 and a leader election protocol has been successfully verified.

1 Introduction

For practical purposes modelling languages are usually extended with the possibility of adding data types and suitable operations. This is for instance done in coloured Petri nets [12] and attributed graph transformation systems (AGTSs) [18, 9]. Extending a GTS with attributes allows one to combine the intuitive graphical aspects of the modelled systems with the natural data structures, which makes such extended GTSs more suitable for practical applications. In some cases attributes can be simulated artificially by encoding them into the graph structure (since GTSs are Turing-complete), but specifying attributes directly leads to more compact models. This is an advantage with respect to over-approximation techniques since we have more control over what is abstracted and in what way it is abstracted.

In the last years we have developed a verification technique for non-attributed graph transformation systems (GTSs) [3], which allowed us to successfully verify several case studies [7, 2, 16]. The technique approximates GTSs by Petri graphs (which are Petri nets with additional hypergraph structure) and refines the obtained Petri graph via counterexample-guided abstraction refinement (CEGAR) when necessary [14]. CEGAR is a standard program analysis technique which refines overly coarse approximations by looking for a spurious run, i.e., a run which violates the property to be verified, but which has no counterpart in the original system. Then the approximation is refined in such a way that the spurious run

* Supported by the DFG project SANDS and CRUI/DAAD VIGONI "Models based on Graph Transformation Systems: Analysis and Verification".

disappears. This procedure can be repeated, but due to the undecidability of the verification problem it is not guaranteed that it will eventually give a definitive yes/no-answer.

In this paper we apply this technique, including abstraction refinement, to AGTSs. We describe our view on AGTSs as graph transformation systems labelled over an algebra and approximate AGTSs by attributed Petri graphs which are basically coloured Petri nets [12] or algebraic high-level nets [8] equipped with a hypergraph structure. After performing the approximation described in [3] attributes are added to the resulting Petri graph, which can then be analyzed as a coloured Petri net. Since the carrier sets underlying data types are often infinite, we additionally need attribute abstraction, which is standard in the framework of abstract interpretation [5]. In the conclusion we will discuss how the approach might be extended to predicate abstraction [10, 11]. The verification technique for AGTSs presented here was implemented in AUGUR 2[1] [15] and we introduce a case study concerning a leader election protocol and describe how it has been verified with AUGUR 2.

2 Attributed Graph Transformation Systems

2.1 Algebras

In this section we describe attributed graph transformation systems (AGTSs). After introducing the (standard) notion of algebra and the (non-standard) notion of Boolean algebra, we show how to define and rewrite attributed graphs.

Definition 1 (signature, algebra). *A* signature Σ *is a pair* $\langle \mathcal{S}, \mathcal{F} \rangle$ *where* $\mathcal{S}$ *is a set of* sorts *and* $\mathcal{F}$ *is a set of function symbols equipped with a mapping* $\sigma\colon \mathcal{F} \to \mathcal{S}^* \times \mathcal{S}$. *Sorts will also be called* types.

A Σ-algebra $\mathcal{A}$ *consists of carrier sets* $(\mathcal{A}_s)_{s\in\mathcal{S}}$ *for each sort and a function* $f^{\mathcal{A}} : \mathcal{A}_{s_1} \times \cdots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$ *for every function symbol* f *with* $\sigma(f) = (s_1 \ldots s_n, s)$.

For a Boolean Σ-algebra *we require that* $\mathcal{S}$ *contains the sort Bool and that we have two subsets* $T_{\mathcal{A}}, F_{\mathcal{A}} \subseteq \mathcal{A}_{Bool}$ *representing the truth values.*

By $T(\Sigma, X)$ *we denote the usual* Σ-term algebra*, where* X *is a set of variables, each equipped with a fixed sort.*

For an algebra $\mathcal{A}$ *we denote by* $\mathcal{A}_{\mathcal{S}}$ *the set* $\mathcal{A}_{\mathcal{S}} = \biguplus_{s\in\mathcal{S}} \mathcal{A}_s$, *i.e., the union of all carrier sets (under the implicit assumption that they are all disjoint).*

Example 1. In our implementation we use an algebra denoted by $\mathcal{C}$ with sorts *Bool*, *Int*, *Str*, *Unit* (which have as carrier sets the standard truth values, integers, strings and one-element set respectively) and tuples over the first three sorts. We consider standard operations, for instance $+, -, *, /$ for the integers and comparison operators $<, \leq, =$ in order to obtain truth values. Operators can also be extended to functions operating on tuples.

We will now define a specific type of algebra needed in the following.

[1] The tool is available at `http://www.ti.inf.uni-due.de/research/augur/`.

Definition 2 (powerset algebra). *For a given Σ-algebra $\mathcal{A}$ we will denote by $\mathcal{P}(\mathcal{A})$ its* powerset algebra *which is an algebra over the same signature. The carrier sets of $\mathcal{P}(\mathcal{A})$ are the powersets of the original carrier sets, i.e., $\mathcal{P}(\mathcal{A})_s = \mathcal{P}(\mathcal{A}_s)$ and function symbols f with $\mathcal{F}(f) = (s_1 \dots s_n, s)$ are interpreted as:*

$$f^{\mathcal{P}(\mathcal{A})}(A_1, \dots, A_n) = \{f^{\mathcal{A}}(a_1, \dots, a_n) \mid a_i \in A_i\},$$

where $A_i \in (\mathcal{P}(\mathcal{A}))_{s_i}$. In the case of a Boolean algebra we set $T_{\mathcal{P}(\mathcal{A})} = \{A' \subseteq \mathcal{A}_{Bool} \mid A' \cap T_{\mathcal{A}} \neq \emptyset\}$ and similarly for $F_{\mathcal{P}(\mathcal{A})}$.

Note that in the case of our example algebra $\mathcal{C}$ we have four truth values in $\mathcal{P}(\mathcal{C})$ where $T_{\mathcal{P}(\mathcal{C})} = \{\{\mathit{true}\}, \{\mathit{true}, \mathit{false}\}\}$, $F_{\mathcal{P}(\mathcal{C})} = \{\{\mathit{false}\}, \{\mathit{true}, \mathit{false}\}\}$. Going to powersets is a necessary step since the concretization of abstract values, which will be introduced later, provides us with an entire set of values, as opposed to a single value. However if we only work with single values, i.e., one-element sets, we will get exactly the same results as in the original algebra.

Finally we need a notion of algebra homomorphism.

Definition 3 (algebra homomorphism). *Let $\mathcal{A}, \mathcal{B}$ be two Σ-algebras. An* algebra homomorphism *$h : \mathcal{A} \to \mathcal{B}$ is a family of maps $(h_s : \mathcal{A}_s \to \mathcal{B}_s)_{s \in \mathcal{S}}$ such that for each $f \in \mathcal{F}$ with $\sigma(f) = (s_1 \dots s_n, s_{n+1})$ we have*

$$h_{s_{n+1}}(f^{\mathcal{A}}(a_1, \dots, a_n)) = f^{\mathcal{B}}(h_{s_1}(a_1), \dots, h_{s_n}(a_n)).$$

2.2 Attributed Graphs

We will now define the notion of graphs we are working with. We consider a fixed set of labels Λ and we start with the definition of hypergraphs and their morphisms.

Definition 4 (hypergraph). *A* hypergraph *G is a tuple (V_G, E_G, c_G, l_G), where V_G is a finite set of nodes, E_G is a finite set of edges, $c_G : E_G \to V_G^*$ is a connection function, and $l_G : E_G \to \Lambda$ is the labeling function.*

We consider a fixed typing function $\mathit{ltype} : \Lambda \to \mathcal{S}$ which associates a sort to each label. The theory could be easily extended to associating several (named) attributes to each label and this is how it is handled in our implementation.

We are now ready to introduce attributed hypergraphs. Note that here we choose a different representation of attributed graphs than in [9] where the focus is on viewing attributed graphs in the framework of adhesive HLR categories and where graphs include specific data nodes. One of our main concerns is to fully separate the graph structure and the attributes for verification purposes.

Definition 5 (attributed hypergraph). *Let $\mathcal{A}$ be a Σ-algebra. An $\mathcal{A}$-*attributed hypergraph *is a tuple $G = (V_G, E_G, c_G, l_G, \mathit{attr}_G)$, where (V_G, E_G, c_G, l_G) is a labelled hypergraph and $\mathit{attr}_G : E_G \to \mathcal{A}_{\mathcal{S}}$ is a function such that for each $e \in E_G$ it holds that $\mathit{attr}_G(e) \in \mathcal{A}_{\mathit{ltype}(l_G(e))}$.*

We consider nodes of a hypergraph as unlabelled (and without attributes). Attributes can be added by providing nodes with unary hyperedges which contain the attribute for that node.

Definition 6 (hypergraph morphisms). *Let G_1, G_2 be two hypergraphs. A* (hypergraph) morphism *$\varphi : G_1 \to G_2$ consists of two total functions $\varphi_V : V_{G_1} \to V_{G_2}$ and $\varphi_E : E_{G_1} \to E_{G_2}$ such that for every $e \in E_{G_1}$ it holds that $l_{G_1}(e) = l_{G_2}(\varphi_E(e))$ and $\varphi_V(c_{G_1}(e)) = c_{G_2}(\varphi_E(e))$. A morphism is called* edge-bijective *(*edge-injective*) whenever it is bijective (injective) on edges. (We will in the following drop the subscripts V and E.)*

Definition 7 (morphisms of attributed hypergraphs). *Let G_1, G_2 be two attributed hypergraphs (where G_1 is attributed over $\mathcal{A}$ and G_2 over $\mathcal{B}$). An* attributed hypergraph morphism *$\varphi = (\psi, h) : G_1 \to G_2$ consists of a hypergraph morphism ψ and an algebra homomorphism $h : \mathcal{A} \to \mathcal{B}$ such that*

$$\forall e \in E_{G_1} : \mathit{attr}_{G_2}(\psi(e)) = h(\mathit{attr}_{G_1}(e)).$$

2.3 Rewriting of Attributed Graphs

Attributed hypergraphs can be transformed using rewriting rules which we define in the following. Our approach follows essentially the presentation in [18], but without using category theory. Furthermore we use the same restrictions on rules as in [3] since this greatly simplifies verification.

Definition 8 (attributed rewriting rule). *We fix a signature Σ and a set X of variables. An* attributed rewriting rule *r is a quadruple (L, R, α, g), where L and R are $T(\Sigma, X)$-attributed hypergraphs, called left-hand side and right-hand side respectively, $\alpha : V_L \to V_R$ is an injective mapping, indicating how nodes are preserved, and $g \in T(\Sigma, X)$ is a guard condition of sort Bool.*

We demand that each of the term attributes of L is a single variable of X (such that each variable appears only once). The set of all variables in the left-hand side is denoted by X'. The right-hand side R may be attributed with arbitrary terms from $T(\Sigma, X')$. Each rule r is associated with a guard expression $g(r) \in T(\Sigma, X')_{Bool}$.

We demand also that there are no isolated nodes in the left-hand side L and no isolated nodes in $V_R - \alpha(V_L)$. Additionally E_L must not be empty and there can not be two edges with the same label in the left-hand side of a rule.

If an instance of the left-hand side is found in the current state of the system, then this rule can be applied and the instance of the left-hand side of the rule will be replaced by its right-hand side. We are now ready to define the notion of attributed graph transformation systems.

Definition 9 (attributed graph transformation system (AGTS)). *An* attributed graph transformation system (AGTS) *$\mathcal{G} = (\mathcal{R}, G_0)$ over an algebra $\mathcal{A}$ is a finite set of attributed rewriting rules $\mathcal{R}$ together with an $\mathcal{A}$-attributed start hypergraph G_0 (also called initial graph).*

We now describe in a set-based notation how rules can be applied to attributed graphs. This could also be done categorically.

Definition 10 (rule application). *A* match *of a rewriting rule* $r = (L, R, \alpha, g)$ *in an* $\mathcal{A}$*-attributed graph* G *is a morphism* $\psi = (\varphi, h) : L \to G$ *which is injective on edges. We can apply* r *to a match in* G *obtaining a new graph* H*, written* $G \overset{r}{\Rightarrow} H$*, whenever the guard expression is satisfied, i.e.,* $h(g) \in T_{\mathcal{A}}$*. 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, labelling and attribute functions are given by*

$$\begin{aligned} e \in E_G - \varphi(E_L) \quad &\Rightarrow \quad c_H(e) = c_G(e), \quad l_H(e) = l_G(e), \quad \mathit{attr}_H(e) = \mathit{attr}_G(e) \\ e \in E_R \quad &\Rightarrow \quad c_H(e) = \overline{\varphi}(c_R(e)), \quad l_H(e) = l_R(e), \quad \mathit{attr}_H(e) = h(\mathit{attr}_R(e)) \end{aligned}$$

That is, a left-hand side is found and replaced by the corresponding right-hand side. We use a restricted version of the DPO (double-pushout) approach where we only allow discrete interfaces. Merging as well as deletion of nodes is forbidden. Edges, however, can be deleted. The new attributes in the right-hand side are obtained by using h, the *binding* of the set of *free variables* X' of the left-hand side.[2]

Example 2. We use the simple AGTS shown in Fig. 1 as a running example. Edges labelled B and C have integer attributes. The attribute in B is increased by one whenever a new edge is created, whereas the attribute in C is multiplied with the corresponding attribute in B when C crosses B. The edges A and Error have no attributes. The property we want to verify is that no Error edge will ever be created. Note that intuitively this holds since no edge labelled 7 will ever be created and hence rule "Cross Backward" will never be applied, since C will always contain a even attribute value.

3 Approximation of Attributes

In Example 1 we considered an algebra with infinite carrier sets. In order to analyse the systems thus obtained we need a mechanism of attribute approximation. Hence we work in the framework of abstract interpretation [5] and start with the notion of a Galois connection, which is basically a pair of adjoints.

Definition 11 (Galois connection on algebras). *Let* $\Sigma = \langle \mathcal{S}, \mathcal{F} \rangle$ *be a signature and let* $\mathcal{A}$*,* $\mathcal{B}$ *be two algebras over this signature, where each carrier set is lattice-ordered via* $\sqsubseteq$*.*[3]

[2] Note that in the case of a powerset algebra some elimination of over-approximation could be useful, by removing attribute values in the right-hand side that did not satisfy the guard expression. In order to be able to represent the theory in a compact way we choose not to follow this path at the moment.

[3] The partial order $\sqsubseteq$ stands for the information ordering. Intuitively whenever $a \sqsubseteq b$, then a is considered to be more exact, i.e., a conveys more information about the system state.

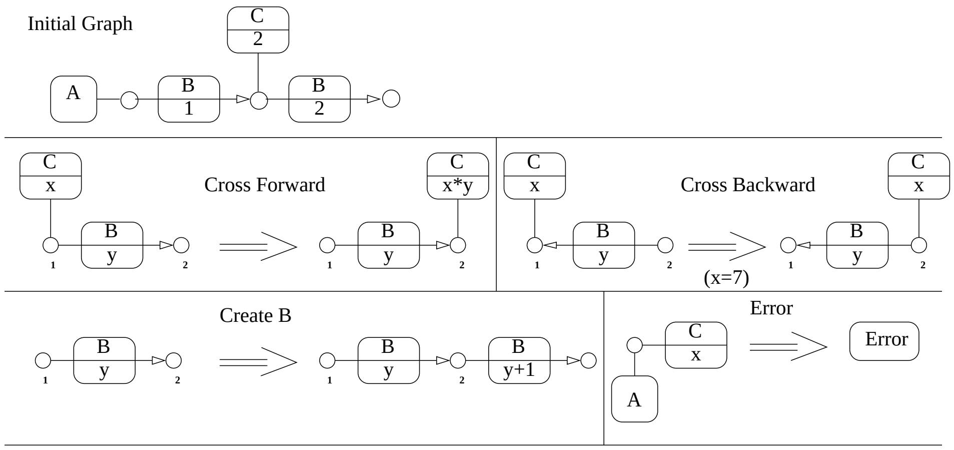

Fig. 1. Example of an attributed graph transformation system

A family of functions $(\alpha_s : \mathcal{A}_s \to \mathcal{B}_s,\ \gamma_s : \mathcal{B}_s \to \mathcal{A}_s)_{s\in\mathcal{S}}$ *is called* Galois connection on algebras *if they are monotone with respect to* $\sqsubseteq$ *and if for all* $s \in \mathcal{S}$*:* $\forall a \in \mathcal{A}_s : a \sqsubseteq \gamma_s(\alpha_s(a))$ *and* $\forall b \in \mathcal{B}_s : \alpha_s(\gamma_s(b)) \sqsubseteq b$.

Finally we require that for each function symbol f *with* $\sigma(f) = (s_1 \dots s_n, s)$ *the function* $f^{\mathcal{B}}$ *is a safe over-approximation of* $f^{\mathcal{A}}$*, i.e., for all* $a_1, \dots, a_n$ *with* $a_i \in \mathcal{A}_{s_i}$ *it holds that:* $\alpha_s(f^{\mathcal{A}}(a_1, \dots, a_n)) \sqsubseteq f^{\mathcal{B}}(\alpha_{s_1}(a_1), \dots, \alpha_{s_n}(a_n))$*. Note that this condition says that* α *is an algebra homomorphism "up to"* $\sqsubseteq$*. Such mappings will also be called* $\sqsubseteq$*-homomorphisms.*

Furthermore if $\mathcal{A}$*,* $\mathcal{B}$ *are Boolean algebras we require that both use the same carrier set for Bool, that* $\alpha_{Bool}, \gamma_{Bool}$ *are identities,* $T_{\mathcal{A}} = T_{\mathcal{B}}$*,* $F_{\mathcal{A}} = F_{\mathcal{B}}$ *and that furthermore truth values respect the information ordering* $\sqsubseteq$ *in the following sense:*

$$\forall v, v' \colon (v' \sqsubseteq v \wedge v' \in T_{\mathcal{A}} \Rightarrow v \in T_{\mathcal{A}}) \qquad \textit{(and similarly for } F_{\mathcal{A}}\textit{).}$$

Example 3. The algebra $\mathcal{C}$ that is used by our implementation allows several possible abstractions via algebras with finite carrier sets, some of which are already predefined. For instance, for the integers we use modulo abstraction modulo base b (each integer k is abstracted by $(k \bmod b)$) and interval abstraction with boundaries m, n (k is abstracted by one of "$< -m$",$-m, \dots, n-1, n$,"$> n$"). We also defined suitable operators on the abstract values which safely over-approximate the original functions in the sense of Definition 11.

However, we can not use directly the algebra $\mathcal{C}$ for a Galois abstraction since it is not lattice-ordered. Hence we work with $\mathcal{P}(\mathcal{C})$ where the lattice-order is set inclusion. Then every set of concrete values is mapped to set of abstract values via α_s, whereas γ_s is the corresponding concretization.

4 Analysis of Attributed Graph Transformation Systems

Since GTSs are in general Turing-powerful, over-approximation techniques are needed for their analysis. In our case we abstract AGTSs by coloured Petri nets, which are a conceptually simpler formalism which is easier to analyse. In [3] an approximated unfolding technique for GTSs was presented, in which—compared to standard unfolding techniques—additional folding steps are used, which over-approximate but guarantee a finite approximation. The resulting over-approximation is a so-called Petri graph which is a Petri net with an additional hypergraph structure, i.e., the hyperedges are at the same time the places of the net. Our idea here is to construct an *attributed* Petri graph which over-approximates an AGTS: an attributed Petri graph consists of an attributed (or coloured) Petri net and a hypergraph structure over it. Our notation is oriented on coloured Petri nets [12] and algebraic high-level nets [8].

4.1 Attributed Petri graphs

We now formally define attributed Petri nets and attributed Petri graphs. We consider a fixed set of labels Λ and a function $ltype : \Lambda \to S$.

By $A^{\oplus}$ we denote the free commutative monoid over A with monoid operation $\oplus$, whose elements are also called *multisets*. A multiset $M \in A^{\oplus}$ can be written as a formal sum $M = \bigoplus_{a \in A} m_a \cdot a$ and given M we write $M(a)$ to denote the coefficient m_a. A function $f : A \to B$ can be extended to a function $f : A^{\oplus} \to B^{\oplus}$ on multisets as follows: For $M \in A^{\oplus}$ we define $M' = f(M)$ with $M'(b) = \sum_{a \in f^{-1}(b)} M(a)$ for every $b \in B$. Besides $\oplus$ we also use *difference* $M \ominus M'$, where $M, M' \in A^{\oplus}$ and *inclusion*, defined by $M \leq M'$, when there exists $M'' \in A^{\oplus}$ such that $M \oplus M'' = M'$

We will now introduce attributed Petri nets which imitate coloured nets [12] in their graphical representation, and which are basically algebraic high-level nets [8] with small variations. For instance, compared to [8], we only allow variables, but not arbitrary terms in the preset of a transition.

Definition 12 (attributed Petri net). *Let $\mathcal{A}$ be a Σ-algebra. An* $\mathcal{A}$-attributed Petri net *is a tuple $\mathcal{N} = (S, T, l, {}^{\bullet}(), ()^{\bullet}, guard, m_0)$, where S is a set of places, T is a set of transitions, $l : S \to \Lambda$ is a labelling function, ${}^{\bullet}(), ()^{\bullet} : T \to (S \to (T(\Sigma, X)_{\mathcal{S}})^{\oplus})$ are pre- and postset functions, $guard : T \to T(\Sigma, X)_{Bool}$ is a guard function, and m_0 is the initial marking of the net. A* marking *of an attributed Petri net is a function $m : S \to \mathcal{A}_{\mathcal{S}}^{\oplus}$. We also require that:*

(1) Each element of the multisets ${}^{\bullet}t(s)$, $t^{\bullet}(s)$ and $m(s)$ is of sort $ltype(l(s))$.
(2) The multiset $\bigoplus_{s \in S} {}^{\bullet}t(s) = X'$ contains only variables, each with multiplicity 1. Furthermore, the elements of $t^{\bullet}(s)$ are contained in $T(\Sigma, X')$ and $guard(t) \in T(\Sigma, X')_{Bool}$.

Elements of $m(s)$ (which are elements of the carrier sets) are also called *tokens*. For a marking m define $|m| : S \to \mathbb{N}$ as $|m|(s) = |m(s)|$, i.e., each place is associated with the number of tokens it contains.

A transition t is *enabled* for the marking m if there exists a binding $h : T(\Sigma, X) \to \mathcal{A}_{\mathcal{S}}$ such that $h(guard(t)) \in T_{\mathcal{A}}$ and for each place s it holds that $m(s) \geq h({}^{\bullet}t(s))$. An enabled transition with a given binding h can be fired and the marking m of the net will be transformed into m', denoted by $m\,[t, h\rangle\, m'$:

$$m'(s) = m(s) \ominus h({}^{\bullet}t(s)) \oplus h(t^{\bullet}(s)).$$

We consider a Petri graph as consisting of an attributed Petri net and a non-attributed hypergraph structure over it.

Definition 13 (attributed Petri graph). *Let $\mathcal{G} = (\mathcal{R}, G_0)$ be an AGTS. An* $\mathcal{A}$-attributed Petri graph *(over $\mathcal{R}$) is a tuple $P = (G, \mathcal{N}, p_N, \mu)$, where G is a (non-attributed) hypergraph, $\mathcal{N}$ is an $\mathcal{A}$-attributed Petri net where the places are the edges of G, p_N associates to each transition t a rule $p_N(t) = (L, R, \alpha, g) \in \mathcal{R}$ such that $guard_P(t) = g$ and μ associates to each transition t from $\mathcal{N}$ with $p_N(t)$ as above a (non-attributed) hypergraph morphism $\mu(t) : L \cup R \to G$ such that ${}^{\bullet}t(s) = \bigoplus_{\mu(t)(e)=s, e \in E_L} attr_L(e)$ and $t^{\bullet}(s) = \bigoplus_{\mu(t)(e)=s, e\in E_R} attr_R(e)$.*

An attributed Petri graph for $\mathcal{G}$ *is a pair (P, ι), where $P = (G, N, p_N, \mu)$ is an attributed Petri graph over $\mathcal{R}$ and $\iota : G_0 \to G$ is a (non-attributed) graph morphism such that $m_0(s) = \bigoplus_{\iota(e)=s} attr_{G_0}(e)$ for each edge $e \in E_{G_0}$.*

Note that the edges of the graph are at the same time the places of the net and that the transitions are labelled with rules of the AGTS.

For each marking m of an attributed Petri graph we define an attributed graph $graph(m)$ as follows: first we take the subgraph G' of G with edge set $E' = \{e \mid m(e) \neq \emptyset\}$ and with all nodes adjacent to some edge in E'. Assume that $m(e) = \bigoplus_{i=1}^{k} a_i$ is the marking of $e \in E'$. Now we replace in G' each e by k edges $e_1, \dots, e_k$ with $l_G(e_i) = l_G(e)$, $c_G(e_i) = c_G(e)$ and $attr_G(e_i) = a_i$.

4.2 Approximated Unfolding

We now describe how to obtain an attributed Petri graph from a given AGTS. First, we unfold the underlying GTS in an approximative way as it is described in [3] without taking attributes into consideration. This is done by starting with the initial graph and applying unfolding steps that "simulate" rule applications by adding transitions, as well as folding steps that merge left-hand sides which are causally dependent. Since the approximated unfolding procedure supplies us with morphisms ι and $\mu(t)$ as described in Definition 13 there is a unique way of adding attributes to the Petri graph after the approximated unfolding. This means that attributes do not affect the unfolding procedure itself in any way.

Still, it is necessary to show that the resulting Petri graph is a valid over-approximation.

Proposition 1. *Let P be an attributed Petri graph for a GTS $\mathcal{G}$ obtained as described above. Then, there exists a simulation relation*[4] *$\mathcal{R}$ between the reachable graphs in $\mathcal{G}$ and the reachable markings in P such that: $(G_0, m_0) \in \mathcal{R}$ and*

[4] In the simulation game every application of a rule r must be answered by a transition labelled r.

for every pair (G, m) there exists an edge-bijective attributed hypergraph morphism (with the identity as algebra homomorphism) $G \to graph(m)$. Specifically this means that every graph reachable in $\mathcal{G}$ is over-approximated by a reachable marking of P.

We extended Augur 2 [15] to construct and analyse over-approximations of AGTSs. Fig. 2 depicts the coarsest over-approximation for the AGTS in Fig. 1 computed by Augur 2. Places, which coincide with the edges, are depicted as boxes with rounded corners, with circle-shaped tokens inside. Transitions are represented by thin black rectangles with guard conditions and preset/postset annotations. For instance $1'x$ on an arc leaving a place means that one token is removed and its value bound to x. Note that the over-approximation below is too coarse since the error edge can be covered. Hence abstraction refinement is necessary.

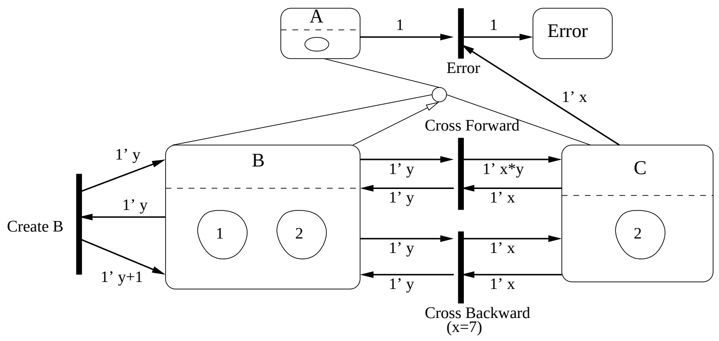

Fig. 2. Petri graph approximating the GTS (first approximation).

4.3 Analysing Petri Graphs

The obtained Petri graphs are basically coloured Petri nets [12] and can be analyzed with techniques developed for such nets. In particular we want to check that certain edges (or places), called error edges, can not be covered. Due to Proposition 1 we can infer that if this holds for the approximation, it is also true for the original system. However, we still have to handle infinite carrier sets, which is done by attribute abstraction. We show here that if the attributes are correctly abstracted, then the abstract version of a Petri graph correctly over-approximates the concrete version.

In the following we assume that AGTSs are attributed over an algebra $\mathcal{A}$, which will be abstracted by an algebra $\mathcal{B}$ via a Galois connection (α_s, γ_s) (see Definition 11). If we take a Petri graph $\mathcal{P}$ attributed over $\mathcal{A}$ this can be easily seen as a Petri graph attributed over $\mathcal{B}$ by applying α_s to all elements of the initial marking. The (abstract) Petri graph obtained in this way is denoted by P^a.

The following proposition shows how the abstract Petri graph P^a can be used in order to analyse P. But let us first fix some notation: For two multisets M_1, M_2 we write $M_1 \sqsubseteq M_2$ if there is a bijection from M_1 to M_2 such that each element of M_1 is smaller than or equal to its image in M_2 (with respect to the information ordering $\sqsubseteq$). For two markings we write $\hat{m}_1 \sqsubseteq \hat{m}_2$ whenever $\hat{m}_1(s) \sqsubseteq \hat{m}_2(s)$ for each place s.

Proposition 2. *For the attributed Petri graphs P and P^a it holds that there is a simulation relation $\mathcal{R}$ on the reachable markings such that $(m_0, m_0^a) \in \mathcal{R}$ and for each pair $(m, \hat{m}) \in \mathcal{R}$ we have $m^a \sqsubseteq \hat{m}$.*

To analyse attributed Petri graphs we need to check whether certain markings or places can be covered by a reachable marking. Hence we adapted two such techniques, coverability graphs [19] and backward reachability [1], to attributed Petri graphs with finite carrier sets and implemented them in AUGUR 2. We also extended both methods to provide us with a trace (= counterexample) to a given coverable marking.

4.4 Abstraction Refinement

This section generalizes the abstraction refinement technique from [14]. We adapt the technique of abstraction refinement for AGTS and attributed Petri graphs. If the analysis of the Petri net gives us a spurious counterexample for the property to verify then we can try to eliminate it using counterexample-guided abstraction refinement [14]. In our case there are two possible ways to refine the obtained over-approximation: either we can refine the graph structure of the obtained over-approximation or the attribute abstraction. One of the challenges is to determine which of the two cases applies.

First we define a notion of (abstract) runs and their correspondence.

Definition 14 (abstract run of an AGTS). *An* abstract run *of an AGTS $(\mathcal{R}, G_0)$ is a sequence of attributed hypergraphs $\mathcal{J} = (J_0 \Rightarrow_{r_1} J_1 \Rightarrow_{r_2} \cdots \Rightarrow_{r_n} J_n)$, where r_i is a rule name, together with (attributed) morphisms $\varphi_i : L_{i+1} \to J_i$ for each $i = 1, \ldots, n-1$, where L_i is the left-hand side of rule $r_i \in \mathcal{R}$.*

Note that we do not demand that J_i can be derived from J_{i-1} by applying rule r_i at match φ_i (hence the name abstract*). If an abstract run is derivable it will be called a* real run*. The j-th prefix of $\mathcal{J}$ is the run $pr_j(\mathcal{J}) = (J_0 \Rightarrow_{r_1} J_1 \Rightarrow_{r_2} \cdots \Rightarrow_{r_j} J_j)$ together with the morphisms φ_i.*

Let $\mathcal{J}' = (J_0' \Rightarrow_{r_1} J_1' \Rightarrow_{r_2} \ldots \Rightarrow_{r_n} J_n')$ be another abstract run with morphisms $\varphi_i' : L_{i+1} \to J_i'$ for each $i = 1, \ldots, n-1$. We say that $\mathcal{J}'$ weakly corresponds *to $\mathcal{J}$ (in symbols $\mathcal{J}' \ll \mathcal{J}$) if for each $i = 1, \ldots, n-1$ there exist edge-bijective (attributed) morphisms $\xi_i : J_i' \to J_i$ for $i = 0, \ldots, n$. If furthermore the following diagram commutes we say that $\mathcal{J}'$* corresponds *to $\mathcal{J}$ and write*

$\mathcal{J}' \lll \mathcal{J}$.

$$\begin{tikzcd} L_{i+1} \arrow[r, "\varphi_i'"] \arrow[rr, bend right, "\varphi_i"'] & J_i' \arrow[r, "\xi_i"] & J_i \end{tikzcd}$$

In both cases, we require that the attributed morphisms are equipped with identity homomorphisms. If they have only $\sqsubseteq$-homomorphisms (as defined in Definition 11) we talk about (weak) $\sqsubseteq$-correspondence *and write $\ll_{\sqsubseteq}$ and $\lll_{\sqsubseteq}$.*

For later use we need following construction (cf. [14]): Let G be a hypergraph and m a marking of the underlying Petri net, specifically $m \in E_G^{\oplus}$. That is, there exists a (non-attributed) morphism $\psi : \mathit{graph}(m) \to G$. Now let $\varphi : G' \to G$ be a morphism such that $\varphi^{\oplus}(E_{G'}) \leq |m|$. Then there exists an edge-injective morphism $e_{m,\varphi} : G' \to \mathit{graph}(m)$ such that $\psi \circ e_{m,\varphi} = \varphi$.

We will mainly use this construction for the special case where $\varphi = \mu(t)|_L : L \to G$, i.e., φ is a match of the left-hand side in the Petri graph (see Definition 13), and m is a marking that allows to fire transition t.

Petri graphs can, as mentioned above, be seen as symbolic representations of graph transition systems and also as representations of sets of abstract runs.

Definition 15 (abstract runs of an attributed Petri graph). *Let (P, ι) with $P = (G, N, p_N, \mu)$ be an attributed Petri graph for an AGTS $(\mathcal{R}, G_0)$. Furthermore let $m_0[t_1, h_1\rangle \dots [t_n, h_n\rangle m_n$ be a firing sequence of the net N and let $r_i = p_N(t_i)$ be the rules corresponding to the transitions. We consider (non-attributed) morphisms $\nu_{i+1} = e_{m_i, \mu(t_{i+1})|_{L_{i+1}}} : L_{i+1} \to \mathit{graph}(m_i)$, where L_{i+1} is the left-hand side of rule r_{i+1} and extend them in the canonical way to attributed morphisms by adding bindings. It is easy to see that the sequence $\mathit{graph}(m_0) \Rightarrow_{r_1} \mathit{graph}(m_1) \Rightarrow_{r_2} \dots \Rightarrow_{r_n} \mathit{graph}(m_n)$ together with the morphisms $\varphi_i = (\nu_i, h_i)$ is an* abstract run.

Each real run $\mathcal{J}_R = (G_0 \Rightarrow_{r_1} G_1 \Rightarrow_{r_2} \dots \Rightarrow_{r_n} G_n)$ of the AGTS $(\mathcal{R}, G_0)$ can be considered as an abstract run where the $\varphi_i : L_{i+1} \to G_i$ represent the matches of the left-hand sides of the rules r_i.

Now let $\mathcal{G}$ be an AGTS, let P be an attributed Petri graph approximating $\mathcal{G}$ and let P^a be the abstract Petri graph derived from P. That is, P over-approximates the (graph) structure, whereas P^a additionally abstracts attributes.

Then, for every real run $\mathcal{J}_R$ of $\mathcal{G}$ there exists an abstract run $\mathcal{J}_A$ of P, such that $\mathcal{J}_R \lll \mathcal{J}_A$. And furthermore for every abstract run $\mathcal{J}_A$ of P there exists an abstract run $\hat{\mathcal{J}}_A$ of P^a such that $\mathcal{J}_A \lll_{\sqsubseteq} \hat{\mathcal{J}}_A$. This is a direct consequence of the simulation property (see Propositions 1 and 2). Since correspondence is transitive this means that every real run $\mathcal{J}_R$ of $\mathcal{G}$ can be associated with an abstract run $\hat{\mathcal{J}}_A$ of P^a such that $\mathcal{J}_R \lll_{\sqsubseteq} \hat{\mathcal{J}}_A$.

We start abstraction refinement with an attributed Petri graph $\mathcal{P}^a$ which is obtained by unfolding an AGTS $\mathcal{G}$ and interpreting the resulting Petri graph in $\mathcal{B}$ (as described in the previous section). If the property we want to verify is violated, we obtain a counterexample of the following form:

$$\hat{m}_0[t_1, \hat{h}_1\rangle \ldots [t_n, \hat{h}_n\rangle \hat{m}_n,$$

where the t_i are transitions and the $\hat{h}_i$ are the corresponding bindings. Usually the AGTSs that we consider have an error rule and the property we want to verify is that this rule is not applicable. Hence an error trace includes a firing of the corresponding error transition as the last step. It can be seen as an abstract run (with abstracted attributes) of the following form:

$$\hat{\mathcal{J}}_A = (\mathit{graph}(\hat{m}_0) \Rightarrow_{r_1} \mathit{graph}(\hat{m}_1) \Rightarrow_{r_2} \ldots \Rightarrow_{r_n} \mathit{graph}(\hat{m}_n)),$$

where $r_j = p_N(t_j)$ and $(\nu_j, \hat{h}_j) : L_j \to \mathit{graph}(\hat{m}_{j-1})$ are the corresponding morphisms from the left-hand side of r_j to $\mathit{graph}(\hat{m}_{j-1})$ for $j = 1, \ldots, n$.

After analysing the Petri graph P^a and searching for counterexamples there could be the following four possibilities:

(1) The property is successfully verified, i.e., no counterexample was found in P^a.

(2) A real (non-spurious) counterexample $\hat{\mathcal{J}}_A$ is found. That is, we have $\mathcal{J}_R \lll_{\sqsubseteq} \hat{\mathcal{J}}_A$ for a real run $\mathcal{J}_R$ of $\mathcal{G}$. In this case we have found an error.

(3) The detected counterexample is *spurious*. This means that no real run $\mathcal{J}_R$ with $\mathcal{J}_R \lll_{\sqsubseteq} \hat{\mathcal{J}}_A$ exists. However, there could be real runs $\mathcal{J}'_R$ shorter than $\hat{\mathcal{J}}_A$ that correspond to a prefix $pr_i(\hat{\mathcal{J}}_A)$ of the counterexample, i.e., $\mathcal{J}'_R \lll_{\sqsubseteq} pr_i(\hat{\mathcal{J}}_A)$. Let k be the maximal length of such a run. The set of all such maximal real runs (there could be several of them) is denoted by $\mathcal{H}$.

For a given $\mathcal{J}'_R \in \mathcal{H}$ there always exists a (unique) run $\mathcal{J}'_A$ of the attributed Petri graph P (with concrete attributes) with morphisms $(\nu_j, h_j) : L_j \to \mathit{graph}(m_{j-1})$ (morphisms ν_j as above) such that $\mathcal{J}'_R \lll \mathcal{J}'_A$ (see Fig. 3). It is easy to see that also $\mathcal{J}'_A \lll_{\sqsubseteq} pr_i(\hat{\mathcal{J}}_A)$.

$$\begin{tikzcd}
\mathcal{J}'_A\colon & \mathit{graph}(m_0) \arrow[r, Rightarrow, "r_1"'] & \mathit{graph}(m_1) \arrow[r, Rightarrow, "r_2"'] & \cdots & \mathit{graph}(m_i) \\
\mathcal{J}'_R\colon & G_0 \arrow[u, "\xi_0"'] \arrow[d, "\hat{\xi}_0"'] \arrow[r, Rightarrow, "r_1"'] & G_1 \arrow[u, "\xi_1"'] \arrow[d, "\hat{\xi}_1"'] \arrow[r, Rightarrow, "r_2"'] & \cdots & G_i \arrow[u, "\xi_i"'] \arrow[d, "\hat{\xi}_i"'] \\
pr_i(\hat{\mathcal{J}}_A)\colon & \mathit{graph}(\hat{m}_0) \arrow[r, Rightarrow, "r_1"'] & \mathit{graph}(\hat{m}_1) \arrow[r, Rightarrow, "r_2"'] & \cdots & \mathit{graph}(\hat{m}_i) \\
 & L_1 \arrow[u, "(\nu_1, \hat{h}_1)"'] \arrow[uuu, bend left, "(\nu_1, h_1)"] \arrow[uu, bend right, "\varphi_0"'] & L_2 \arrow[u, "(\nu_2, \hat{h}_2)"'] \arrow[uuu, bend left, "(\nu_2, h_2)"] \arrow[uu, bend right, "\varphi_1"'] & &
\end{tikzcd}$$

Fig. 3. Counterexample (abstract and real runs with corresponding left-hand sides)

We now distinguish the following two cases:

(3a) We say that the over-approximation is *structurally too coarse* if for some $\mathcal{J}'_R \in \mathcal{H}$ the corresponding run $\mathcal{J}'_A$ can be extended to a run $\mathcal{J}''_A$ of length

$k+1$ with a morphism $(\nu_{k+1}, h_{k+1}) : L_{k+1} \to \mathit{graph}(m_k)$ in such a way that $\mathcal{J}''_A \lll_{\sqsubseteq} pr_{k+1}(\hat{\mathcal{J}}_A)$. The set of such prefixes of $\mathcal{H}$ is denoted by $\mathcal{H}_S$. Below we describe a technique based on the one proposed in [14] which allows us to eliminate the obtained counterexample in this case.

(3b) In the last case for each run $\mathcal{J}'_R \in \mathcal{H}$ such that $\mathcal{J}'_R \lll_{\sqsubseteq} pr_k(\hat{\mathcal{J}}_A)$, the corresponding run $\mathcal{J}'_A$ can not be extended as in the previous case, i.e., $\mathcal{H}_S = \emptyset$. If this holds then we say that in the over-approximation P^a *the attribute abstraction is too coarse.*

In the first two cases we have solved the problem either with a positive or a negative outcome. If the obtained over-approximation is structurally too coarse (case (3a)) and does not allow us to verify the property, a counterexample-guided abstraction refinement technique [14] for refining the approximation is available. It uses the set $\mathcal{H}_S$ of prefixes of the counterexample and refines the structure of the Petri graph. This is done by identifying which nodes have previously been merged or folded erroneously and by restarting the approximated unfolding from scratch, but making sure that those node are now kept separate. The technique described in [14] for non-attributed GTSs can be applied here without modification. As in [14], we will eliminate not only the spurious run $\hat{\mathcal{J}}_A$ but all other abstract runs corresponding to it and at the same time having a weak correspondence to some run in $\mathcal{H}_S$.

Proposition 3. *The structurally refined Petri graph P'^a constructed above does not contain any run $\hat{\mathcal{J}}'_A$ which $\sqsubseteq$-corresponds to the spurious run $\hat{\mathcal{J}}_A$ of P^a and has a weak $\sqsubseteq$-correspondence to some run in $\mathcal{H}_S$. Furthermore if P'^a contains a spurious run $\hat{\mathcal{J}}'_A$, then it $\sqsubseteq$-corresponds to some run $\hat{\mathcal{J}}_A$ in P^a.*

To refine the approximation in the last case we make our abstraction of attributes more exact in a predefined way. For example for the modulo abstraction we can increase the modulo base b (we usually multiply it by two), and for the interval abstraction we can increase the interval bounds m and/or n. However in this case we have no guarantee that the spurious counterexample will be eliminated. In the implementation the attribute abstraction is refined a predefined number of times and if spurious counterexample are then still reproducible, we terminate with the answer "don't know". Future work is the integration of the predicate abstraction which will be discussed in the conclusion.

So our results for the refinement of attribute abstraction are weaker than in the case of structure refinement. But we can still show that whenever we refine the attribute abstraction in a certain way, no new spurious runs will appear.

Proposition 4. *Let $(\alpha_s : \mathcal{A}_s \to \mathcal{B}_s,\ \gamma_s : \mathcal{B}_s \to \mathcal{A}_s)_{s\in\mathcal{S}}$ be the Galois connection between algebras $\mathcal{A}$, $\mathcal{B}$ which was originally used for attribute abstraction. Now let $(\alpha'_s : \mathcal{A}_s \to \mathcal{D}_s,\ \gamma'_s : \mathcal{D}_s \to \mathcal{A}_s)_{s\in\mathcal{S}}$ be a new connection from $\mathcal{A}$ to $\mathcal{D}$.*

We furthermore assume that there exists a Galois connection from $\mathcal{D}$ to $\mathcal{B}$ with mappings α''_s, γ''_s such that $\alpha_s \sqsupseteq \alpha''_s \circ \alpha'_s$. Then if the refined Petri graph P'^a contains a run $\hat{\mathcal{J}}'_A$, it $\sqsubseteq$-corresponds (with α''_s as $\sqsubseteq$-homomorphisms) to some run $\hat{\mathcal{J}}_A$ in P^a. In particular, if $\hat{\mathcal{J}}'_A$ leads to a marking covering an error edge, then the same is true for $\hat{\mathcal{J}}_A$.

We can iterate abstraction refinement by storing an arbitrary number of spurious counterexamples. Naturally, due to undecidability and the fact that AGTSs are in general Turing-complete, there is no guarantee that this loop will ever terminate.

Example 4. Let us now consider the Petri graph in Fig. 2 using a modulo abstraction with base one (unit abstraction). The edge labelled Error of the Petri graph can be covered by firing transition “Error”. This means that either the property does not hold or the over-approximation is too coarse. In this case one can show that the run is spurious, i.e., it has no counterpart in the original AGTS and the over-approximation is structurally too coarse (case (3a)). Applying abstraction refinement gives us a refined Petri graph (which is not depicted here due to space constraints).

Now an error edge is still in the approximation and a counterexample can be constructed (via rules “Cross Backward”, “Error”). However, this counterexample can not be reproduced without attribute approximation, which means that the abstraction is too coarse and should be refined (case (3b)). By using base two in the modulo abstraction we obtain a Petri graph in which the Error-edge is no longer coverable, which means successful verification.

5 Example: Leader Election

In this section we sketch the modelling and verification of a leader election protocol in a ring architecture with AGTSs. The purpose of the protocol is to elect a unique leader among the stations in a ring-shaped network.

The algorithm uses only local communication and does not depend on the size of the ring. The leader is chosen on the basis of the unique ids of the stations and will eventually be the station with the smallest id (in our case: id 1). Each station sends a message with its id around the ring. Upon reception of the message from its predecessor a station compares the received id with its own id and, if the incoming id is smaller than its own, forwards the message. If the id is larger, then the message is discarded. If a station receives its own id, then it declares itself the leader. The protocol is parametrized in the sense that we can create rings of arbitrary size with a potentially infinite number of stations. We want to show that we never choose the wrong leader, i.e., there is never a situation where we have a station that has a smaller id than the current leader.

Concerning the ids we use interval abstraction with start interval $[0, 1]$. After unfolding the AGTS and analysing it via the coverability technique we obtain a spurious counterexample. Afterwards three iterations of abstraction refinement can be applied: two with structural refinement and one with attribute abstraction in the interval $[0, 2]$. The coverability check then shows that we have successfully verified the protocol. The whole verification procedure for the leader election protocol took 48.15 seconds.[5] More details on this case study will be reported in [17].

[5] All experiments were made using Augur 2 [15] written in C++ under Linux and the computer parameters are 2*Genuine Intel(R) 1.66 GHz with 2.0 GB RAM.

6 Conclusion

We have presented a framework for the verification of attributed graph transformation systems, using approximated unfoldings, attribute abstraction and a counterexample-guided abstraction refinement technique.

There are some related approaches to the verification of graph transformation systems in the literature, see for instance [20, 21, 6, 4]. However, there seems to be only a small amount of work on the verification and over-approximation of attributed graph transformation systems. We are currently aware of attributes in the tool Groove [13] for the verification of finite-state graph transformation systems. Furthermore AGTSs could be transformed into the input language of more conventional model checkers that do support attributes.

This combination is clearly of practical interest and also raises interesting methodological questions. As we have shown the combination of structural refinements and refinements of attribute abstractions is non-trivial.

Currently we handle the refinement of attribute abstraction semi-automatically, by leaving the choice mainly to the user. Clearly this is not completely satisfactory. A natural question to ask is whether the counterexample-guided abstraction refinement approach based on predicate abstraction and Craig interpolation [10, 11] can be adapted to our setting. In this approach the abstraction is refined by generating new predicates over the program variables, based on the counterexample. In our setting the difficulty is not so much how to generate these predicates (after all, we have a specific counterexample) but how to interpret them over the markings of the Petri net. The situation would be easy if all predicates were unary, since in this case we would employ the concept of Galois connections introduced in this article. However generated predicates typically have a higher arity, often predicates are binary predicates of the form $\mathtt{x} < \mathtt{y}$. For the original predicate abstraction approach this is not a problem since there are only finitely many variables and the value of predicates for an abstract state can be described in a finite way. However in our case there can be arbitrarily many tokens and it is not clear to us how to solve the coverability problem for Petri nets with such an abstraction mechanism.

In addition we need more (and larger) case studies in order to test our techniques. Currently we are working on the verification of variants of the Needham-Schroeder protocol a cryptographic protocol used for authentication (see [17]).

References

1. P. Aziz Abdulla, B. Jonsson, M. Kindahl, and D. Peled. A general approach to partial order reductions in symbolic verification. In *Proc. of CAV '98*, pages 379–390. Springer, 1998. LNCS 1427.
2. P. Baldan, A. Corradini, J. Esparza, T. Heindel, B. König, and V. Kozioura. Verifying red-black trees. In *Proc. of COSMICAH '05*, 2005. Proceedings available as report RR-05-04 (Queen Mary, University of London).
3. P. Baldan, A. Corradini, and B. König. A static analysis technique for graph transformation systems. In *Proc. of CONCUR '01*, pages 381–395. Springer-Verlag, 2001. LNCS 2154.

4. J. Bauer and R. Wilhelm. Static analysis of dynamic communication systems by partner abstraction. In *Proc. of SAS '07*, pages 249–264. Springer, 2007. LNCS 4634.
5. P. Cousot. *Abstract Interpretation*. ACM Computing Surveys, 1996.
6. F.L. Dotti, L. Foss, L. Ribeiro, and O. Marchi Santos. Verification of distributed object-based systems. In *Proc. of FMOODS '03*, pages 261–275. Springer, 2003. LNCS 2884.
7. F.L. Dotti, B. König, O. Marchi Santos, and L. Ribeiro. A case study: Verifying a mutual exclusion protocol with process creation using graph transformation systems. Technical Report 08/2004, Universität Stuttgart, 2004.
8. H. Ehrig, J. Padberg, and L. Ribeiro. Algebraic high-level nets: Petri nets revisited. In *Selected papers from the 9th Workshop on Specification of Abstract Data Types '92*, pages 188–206. Springer, 1994. LNCS 785.
9. H. Ehrig, U. Prange, and G. Taentzer. Fundamental theory for typed attributed graph transformation. In *Proc. of ICGT '04*, pages 161–177. Springer, 2004. LNCS 3256.
10. S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *Proc. of CAV'97*, pages 72–83, 1997.
11. T.A. Henzinger, R. Jhala, R. Majumdar, and K.L. McMillan. Abstractions from proofs. In *Proc. of POPL'04*, pages 232–244. ACM, 2004.
12. K. Jensen. Coloured Petri nets: Status and outlook. In *ICATPN*, pages 1–2, 2003.
13. H. Kastenberg. Towards attributed graphs in GROOVE. In *Proceedings of Workshop on Graph Transformation for Verification and Concurrency*, volume 05-34 of *CTIT Technical Report*, pages 91–98, 2005.
14. B. König and V. Kozioura. Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In *Proc. of TACAS '06*, pages 197–211. Springer, 2006. LNCS 3920.
15. B. König and V. Kozioura. Augur 2—a new version of a tool for the analysis of graph transformation systems. In *Proc. of GT-VMT '06 (Workshop on Graph Transformation and Visual Modeling Techniques)*, volume 211 of *ENTCS*, pages 201–210. Elsevier, 2008.
16. V. Kozioura. Verification of random graph transformation systems. In *Proc. of GT-VC '06 (Graph Transformation for Verification and Concurrency)*, volume 175.4 of *ENTCS*, 2006.
17. V. Kozyura. *Abstraction and Abstraction Refinement in the Verification of Graph Transformation Systems*. PhD thesis, Universität Duisburg-Essen, forthcoming.
18. M. Löwe, M. Korff, and A. Wagner. An algebraic framework for the transformation of attributed graphs. In *Term graph rewriting: theory and practice*, pages 185–199. John Wiley and Sons Ltd., 1993.
19. W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1985.
20. A. Rensink and D. Distefano. Abstract graph transformation. In *Proc. of SVV '05 (3rd International Workshop on Software Verification and Validation)*, volume 157.1 of *ENTCS*, pages 39–59, 2005.
21. D. Varró. Towards symbolic analysis of visual modeling languages. In *Workshop on Graph Transformation and Visual Modeling Techniques '02*, volume 72 of *ENTCS*. Elsevier, 2002.