Using Attributed Graph Grammars to Verify Properties of a Mobile Internet Protocol

Simone André da Costa[1,2] and Leila Ribeiro[1⋆] and
Fernando Luís Dotti[4⋆⋆] and Antônio Carlos da Rocha Costa[1,3]

[1] Universidade Federal do Rio Grande do Sul, INF, Brazil
[2] Universidade Federal de Pelotas, Dept. Informática, Brazil
[3] Universidade Católica de Pelotas, Escola de Informática, Brazil
[4] Pontifícia Universidade Católica do Rio Grande do Sul, FACIN, Brazil

Abstract. Graph grammar is a formal language that is suitable for the description of distributed systems, and is intuitive even for non-theoreticians. The extension of graph grammars to attributed graphs gives raise to a framework to reason about attributes (data values). We use a relational and logical approach for graph grammars, and use mathematical induction to analyze properties of the specifications, enabling the proof of properties for systems with an infinite state-space and infinite computations. Proofs by induction can be semi-automated by using theorem provers. Two important questions of a user when using theorem provers to analyze a system are: (1) how to state the desired properties; and (2) which kind of help must be given to complete the proof of the property. In this paper, we tackle these two problems, illustrating our approach with an example: the mobile support for Internet Protocol version 6 (IPv6). This protocol defines how mobile devices can move from one network to another while maintaining ongoing communication.

1 Introduction

Graph grammars is a specification language that allows the description of a system in terms of states and state changes, where the states are described by graphs and the state changes are described by rules having graphs at the left- and right-hand sides. The operational behavior of the system is expressed via applications of the rules to the state-graphs depicting the actual states of the system. The graphical notation and the definition of local changes to system state using rules make it suitable for describing distributed systems [1, 2].

Attributed graph grammars enrich the graph grammar formalism integrating data types into graphs, by allowing assignment of values to vertices and/or edges. It is possible to use variables and terms in rules, giving the specifier a better level of abstraction with respect to grammars using only non-attributed graphs. For verification, however, the presence of attributes adds some difficulties, since data

⋆ Supported by CNPq/Brazil
⋆⋆ Supported by CNPq/Brazil grant 200806/2008-4

types typically involve infinite sets of values. There are approaches to deal with verification of infinite state attributed graph grammars [3, 4], but their analysis process is for limited classes of grammars.

In previous works [5, 6], we presented a relational and logical approach for attributed graph grammars. These works provided the basis for the verification of a large class of (infinite-state) graph grammars through mathematical induction. Proofs by induction can be semi-automated by using theorem provers. In contrast to another established techniques of verification, such as model-checking [7, 8], theorem proving allows the analysis of a infinite-state system without any kind of approximation. On the other hand, the adoption of such technique may require interaction with the system developer.

Two important questions of a user when using theorem provers to analyze a system are: (1) how to state the desired properties; and (2) which kind of help must be given to complete the proof of the property. In this paper, we tackle these two problems, by explaining how to state properties and construct corresponding proofs, and illustrating our approach with an example: the mobile support for Internet Protocol version 6 (IPv6). This protocol defines how mobile devices can move from one network to another while maintaining ongoing communication.

The paper is organized as follows. Section 2 reviews attributed graph grammars. Section 3 specifies the mobile support for Internet protocols. Verification of properties is discussed in Section 4. Section 5 presents conclusions.

2 Attributed Graph Grammars

Graph-based formal description techniques often present a friendly means of explain complex situations in a compact and understandable way. Graph grammars generalize Chomsky grammars from strings to graphs [9]. The basic idea of this formalism consists on specifying the states of a system as graphs and describing the possible state changes as rules, having graphs at the left- and right-hand sides. Graph rules are used to capture the dynamical aspects of the systems. That is, from the initial state of the system (the initial graph), the application of rules successively changes the system state.

Attributed graph grammars [4] are an extension to the basic formalism integrating the use of data types into graphs. An attributed graph has two components: a graphical part (a graph composed of vertices and edges) and a data part. The use of attributed graphs gives to the specifier a language that is more suitable for specification, merging the advantages of the graphical representation with the standard representation of classical data types. In our approach, the graphical and the data part are linked by attribution functions. That is, a graph has some special kind of edges called *attribute edges* that are used to describe attribution of vertices, and a mathematical function assigns a data value to each one of these attribute edges. We use algebraic specifications to define data types and algebras to describe the values that can be used as attributes.

An attributed graph grammar is a tuple consisting of a *type graph*, an *initial graph* and a *set of rules*. The *type graph* characterizes the types of vertices and

edges allowed in the system. The part of the type graph describing data elements consists of names of types. The *initial graph* represents the initial state of the system and the *set of rules* describes the possible state changes that can occur. A rule has a left-hand side (LHS) and a right-hand side (RHS), which are both graphs, and a partial graph morphism that connects these graphs in a compatible way and determines what should be modified by the rule applications. Attributes in LHS and RHS of rules must be variables and the possible relations between these variables are expressed by equations associated to each rule.

Intuitively, a state change occurs by applying a rule to a graph. A rule is applicable in a graph if there is a match, that is, an image of the left-hand side of the rule in the graph. Roughly speaking, this means that all items (vertices and edges) belonging to the left-hand side must be present at the current state and all variables of the left-hand side must be assigned to actual values of the current state to allow a rule to be applied (also all rule equations and equations of the specification must be satisfied by the chosen assignment of values to the variables). Each rule application transforms a state-graph by replacing a part of it by another graph. This process occurs in the following way: all items mapped from the left to the right-hand side (via the graph morphism) must be preserved; all items not mapped must be deleted from de current state; and all items present in the right-hand side but not in the left-hand side must be added to the current state to obtain the next one. The values of the attribute edges that came from the current state and are not in the left-hand side of the rule are preserved while the values of the preserved or added attribute edges are determined by the equations of the rule and by the assignment of the values to the variables of the right-hand side of the rule.

In previous work [6], we have defined a translation of attributed graph grammars to relational structures. Given an attributed graph grammar, we associate to it a tuple composed of a set and a collection of relations. The set describes the domain of the structure (the set of vertices, edges and attribute values of the graph grammar) and the relations define the type graph, the initial graph and the rules. For instance, relations $vert_G$ (unary), inc_G (ternary) and $attr_G$ (binary) model an attributed graph:

- $vert_G$ defines the set of vertices of G;
- inc_G represents the incidence relation between vertices and edges of G;
- $attr_G$ specifies the values of the attribute edges of G.

If we consider the graph depicted in Figure 1, we have for example, $\{\mathsf{MN}, \mathsf{R}, \mathsf{bindAck}\} \subset vert_G$, $\{(\mathsf{homeAg}, \mathsf{MN}, \mathsf{R}), (\mathsf{con}, \mathsf{R}, \mathsf{MN})\} \subset inc_G$ and $\{(\mathsf{addr}, Nat), (\mathsf{bindCache}, Cache)\} \subset attr_G$. A series of logical conditions impose restrictions to the elements of such relations in order that they really represent the components of an attributed graph grammar (graphs, typed graphs, graph morphisms and rules). Details can be found in [6]. In this approach, the application of a rule is described by a definable transduction (that can be seen as an inference rule) on the relational structure associated to a graph grammar.

3 Example: Mobile IP

Originally, the design of the Internet protocol suite did not account for the possibility of mobile nodes. Portable computers and several IP capable devices, as well as wireless connectivity, became very common in the last decade and the requirement to support communication transparent node mobility became thus natural: a mobile node should be able to dynamically change the access network while transparently maintaining ongoing communication at application level. This is intrinsically a routing problem and is solved through similar extensions of both IPv4 and IPv6 protocols: on top of existing IP functionality, support mobile nodes by dynamically assigning IP addresses in the networks they enter and redirecting existing traffic to the current location (address) of the mobile node. In the following, mobile IPv6 [10] explanations are introduced together with their formalization in Graph Grammars.

3.1 Graph and Data Types

For this example, we use the algebraic specification **MobileIP**, consisting of basic data types like natural numbers (*Nat*), booleans (*Bool*), pairs (*Address*), and lists of numbers (*ListNat*), lists of pairs (*ListAdd*), pairs of a number and an address (*Cache*) and usual operations. This specification is omitted due to space limitations. The types of elements of the example can be seen in Figure 1. There are 3 kinds of vertices: MN (mobile node), R (router) and message vertices, corresponding to mobile nodes, routers and messages, respectively. A Mobile Node (MN) is connected to its home agent (homeAg - responsible to forward data packets to MN while away from home), and to a default router (default), both of type R (router). A Mobile Node has a home address[5] (homeAd of type *Address*) in its home network. When a MN visits a foreign network, it may obtain an address in that network, called Care-of Address (CoA), together with a router address in that network. This pair is stored in bindUpdL (of type *ListAdd*, recording pairs care of address /router address). Several access networks may be available in a given location. MN can hold several such pairs. At any time away from home, an MN is using one primaryCoA (of type *Address*) to be addressed, as well as one default router, respective to the CoA in use. An MN may be in a handover (of type *Bool*) activity, whereby the primaryCoA and default router are changed according to the chosen access network.

Both the home agent and the current default router of a mobile node are of router type R. Each R vertex has an addr (of type *Nat*, the router's address). It supports address distribution to nodes, the address space managed by the router is represented by freeAddrL (of type *ListNat*, recording the free addresses of this router). As home agent, the router has to record the current CoA of the mobile nodes belonging to its home network, supported by bindCache (of type *Cache*).

[5] In the graphical representation, attribute edges are drawn as dashed edges connecting a vertex and a data type. The same data types appear multiple times for convenience (otherwise the graphical representation would be too cumbersome).

This allows the home agent to forward data packets to the current location of the mobile node.

There is one message vertex for each type of message that can be sent in this system. This message vertex is connected to a target node (that may be : MN or R, depending on the message), and may have different types of arguments. The use of messages is explained with the rules.

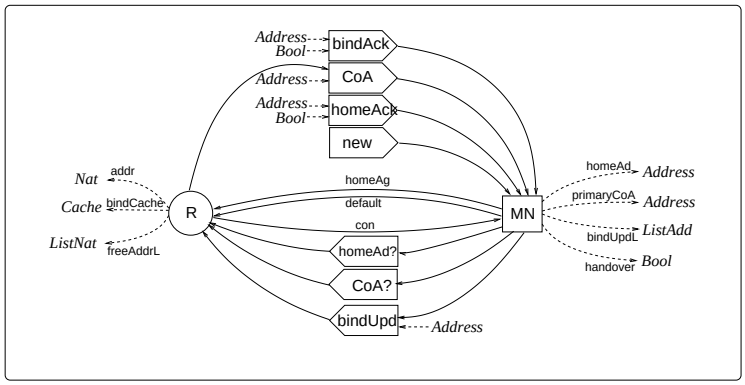

Fig. 1. Type Graph

3.2 Graph Grammar for Mobile IP

Figures 2 and 3 present the rules. Mobile Nodes can be dynamically created (rule $createMN$) and initialized by asking to a reachable router for their home address (rule $home?$). A router may respond positively, reserving an address (rule $ackHome$) or negatively (rule $nackHome$) in case, for instance, there is no address available. While moving away from home, mobile nodes may reach other routers and ask for care-of-addresses (rule $reqCoA$). If accepted in the foreign network, a pair of care-of-address and router address are returned (rule $recCoA$). When moving, a mobile node has to select among the CoAs, which is to be used as primary, modifying the default router accordingly and updating this information in the home agent. This comprises a handover procedure and is initiated by a mobile node (rule $reqBind$). The home agent responds to the update request positively (rule $bind$) or negatively (rule $notBind$), finishing the handover procedure. In our modelling a router reacts to mobile node requests to: obtain a home address (rules $reqHome1$ and $reqHome2$); to obtain a care-of-address (rule $respCoA$); and to update the binding of a mobile node to a care-of-address (rules $bindOk$ and $bindNOK$).

4 Verification of Properties

The relational and logical approach for attributed graph grammars [5, 6] allowed the use of mathematical induction to verify properties for systems with an infinite state-space and infinite computations. The verification process involves two main steps: the statement of properties and the proof of properties.

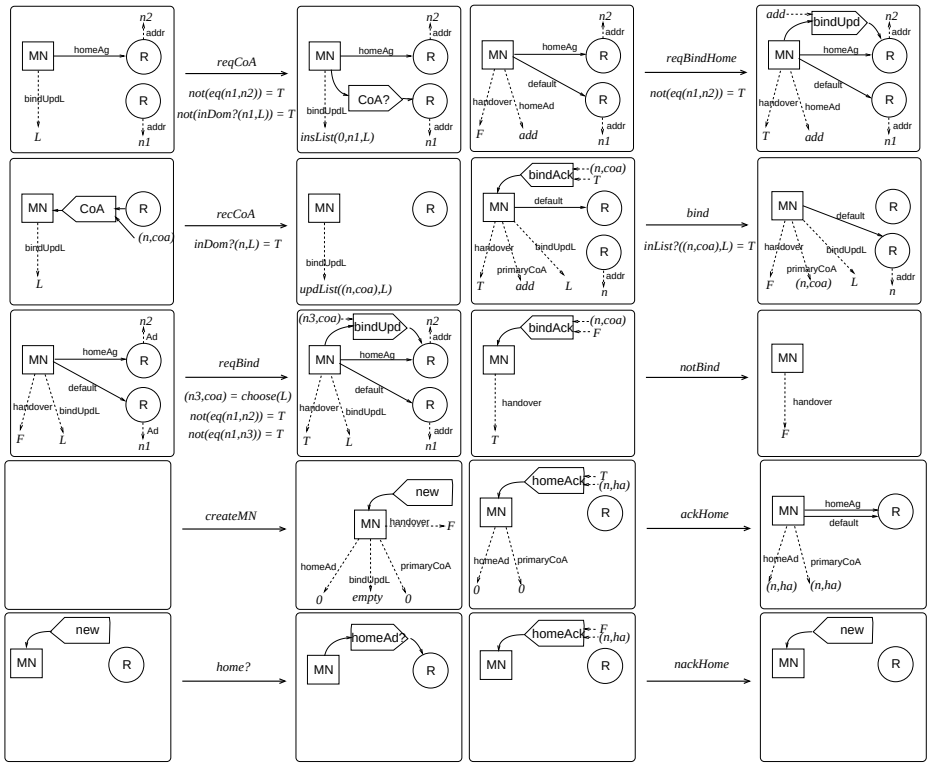

Fig. 2. Mobile Node Rules

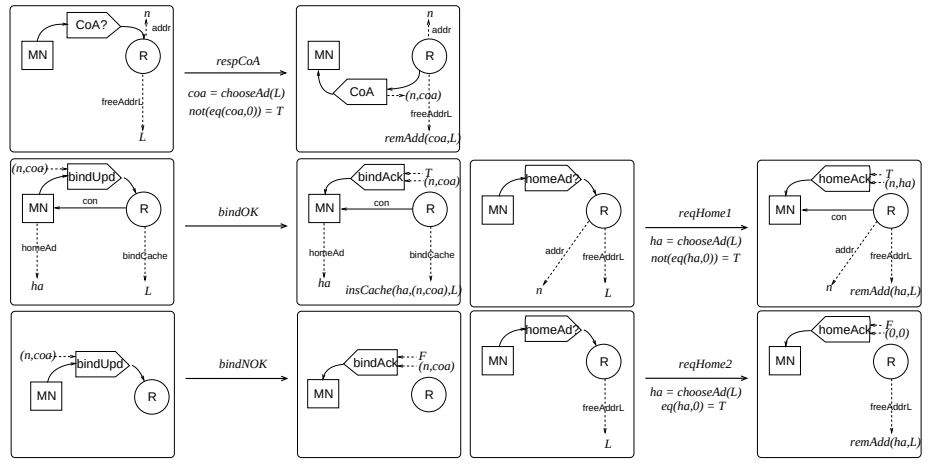

Fig. 3. Router Rules

4.1 Stating Properties

The first step in the verification process consists on the specification of system requirements. These descriptions must be precise and unambiguous to enable verification through (semi-)automated tools. In our approach we use first-order logic. Although we are dealing with a very wide spread mathematical language, some mathematical expertise is required to write correctly the specifications.

In order to help and simplify this task, we defined patterns for property specifications [6]. The patterns use pre-defined functions, that are recursively defined: each function is specified for the initial graph ($G0$) as the base case and for any graph resulting of a rule application ($ap_m^{\alpha i}G$, graph resulting from the application of rule αi at match m) in the inductive step. Examples of such functions are given in Table 1. Function edg_{t_1} returns the edges of a reachable graph of type t_1, function edg returns all edges with their respective source and target vertices of a reachable graph and function att_E returns all pairs of attribute edges and respective values of a reachable graph.

Table 1. Functions in the Standard Library

Description	Function Definition
Edges of specific type	$edg_{t_1}\ G0 = \{x \mid t_E^{G0}(x,t_1)\}$ $edg_{t_1}\ ap_m^{\alpha i}G = \{x \mid t_E^{Ri}(x,t_1) \vee [x \in edg_{t_1}G\ \wedge \nexists w\ m_E^{\alpha i}(w,x)]\}$
Edges with source and target vertices	$edg\ G0 = \{(x,y,z) \mid inc_{G0}(x,y,z)\}$ $edg\ ap_m^{\alpha i}G = \{(x,y,z) \mid [(x,y,z) \in inc\ G \wedge \nexists w\ m_E^{\alpha i}(w,x)] \vee$ $\vee\ [\exists r,s\ inc_{Ri}(x,r,s) \wedge \exists w_1,w_2\ [\alpha i_V(w_1,r) \wedge$ $\wedge\ \alpha i_V(w_2,s) \wedge m_V^{\alpha i}(w_1,y) \wedge m_V^{\alpha i}(w_2,z)]] \vee$ $\vee\ [inc_{Ri}(x,y,z) \wedge \nexists w_1,w_2\ [\alpha i_V(w_1,y) \wedge$ $\wedge\ \alpha i_V(w_2,z)]] \vee \exists r\ [inc_{Ri}(x,r,z) \wedge$ $\wedge\ \exists w_1\ [\alpha i_V(w_1,r) \wedge m_V^{\alpha i}(r,y)] \wedge$ $\wedge\ \nexists w_2\ \alpha i_V(w_2,z)] \vee \exists s\ [inc_{Ri}(x,y,s) \wedge$ $\wedge\ \nexists w_1\ \alpha i_V(w_1,y) \wedge \exists w_2[\alpha i_V(w_2,s) \wedge m_V^{\alpha i}(s,z)]]\}$
Attributes of edges	$att_E\ G0 = \{(x,a) \mid attr_{G0}(x,a)\}$ $att_E\ ap_m^{\alpha i}G = \{(x,a) \mid attr_{Ri}(x,a) \vee [(x,a) \in att_E\ G \wedge$ $\wedge\ \nexists w\ m_E^{\alpha i}(w,x)]\}$

Relations used in definitions are part of the relational approach for graph grammars. For instance, t_E^G is a binary relation that associates the edges of graph G with their types. Relation αi_V is the binary relation that maps the vertices of the left-hand side (Li) of the rule αi to vertices of its right-hand side (Ri). The pair of binary relations $(m_V^{\alpha i}, m_E^{\alpha i})$ defines a match. All relations of the relational structure that characterizes the graph grammar are considered as axioms, that is, considering R a relation belonging to the relational structure, $R(x_1,\ldots,x_n) \equiv true$ iff $(x_1,\ldots,x_n) \in R$.

Using the pre-defined functions, a collection of patterns has been defined [6]. Examples are described in Table 2. Once the developer identifies requirements of a system, patterns that fit those requirements are selected and instantiated.

Considering the Mobile IP example, the pattern system and the standard library can assist, for example in the statement of the following properties.

Table 2. Properties Patterns

Property	**Pattern**
If there is an edge of type t_1, then there is no edge of type t_2 with the same source vertex.	$\exists x_1, y_1, z_1 [x_1 \in edg_{t_1} g \wedge (x_1, y_1, z_1) \in edg\ g] \rightarrow \nexists x_2, y_2\ [x_2 \in edg_{t_2} g \wedge (x_2, y_1, y_2) \in edg\ g]$
For all edges of type t, its target vertex has an attribute of type t_2 with the same value of as the attribute of type t_1 of its source vertex.	$\forall x_1, y_1, z_1 [x_1 \in edg_t g \wedge (x_1, y_1, z_1) \in edg\ g] \rightarrow \exists x_2, x_3, n [x_2 \in edg_{t_1} g \wedge (x_2, y_1, y_1) \in edg\ g \wedge (x_2, n) \in att_E g \wedge x_3 \in edg_{t_2} g \wedge (x_3, z_1, z_1) \in edg\ g \wedge (x_3, n) \in att_E g]$

Description Each mobile node has only one home agent.
Property 1 If there is an edge of type homeAg, then there is no other edge of type homeAg with the same source vertex.
Pattern first of Table 2.

Description A mobile node is always connected through its primary CoA.
Property 2 For all edges of type default, its target vertex has an attribute of type addr with the same value of as the first component of the attribute of type primaryCoA of its source vertex.
Pattern second of Table 2.

4.2 Proving Properties

The relational approach of graph grammars has been developed to allow the use of theorem provers, short TP, for verification of properties. In contrast to other methods of verification, like model checking [7], theorem proving allows the application of techniques such as structural induction to prove properties over infinite domains. Nevertheless, the adoption of such technique may require interaction with the system developer, and thus, the user may have some knowledge of how proofs are constructed to interact correctly with the system.

The proof strategy used to verify properties for systems specified as graph grammars in a theorem prover is the following. First, we define the relational structure associated to the graph grammar (this can be totally automated using the definitions in [6]). The relations of this structure define axioms that can be used in proofs. Then, we state the property to be proven using logic formulas (possibly with aid of property patterns). Finally, the proof is constructed. Properties about reachable states are proven by induction: in the base case, the property is verified for the initial graph (this step can be totally automated, since it involves only consulting the relations that define the initial graph) and then, for the inductive step, the property is verified for each rule applicable to a reachable graph. The proof process for this step may be semi-automated, that is, it may proceed until an auxiliary theorem is required; in this case, this theorem must be proven before resuming the proof of the original goal. In many cases, the proof of a property may depend on the establishment of a set of other properties or theorems. However, it is important to notice that all these auxiliary theorems can be used as simplification rules, and will probably be reused in future proofs. Next, we describe proof schemes for the properties stated in the previous section.

Property 1 *If there is an edge of type* homeAg*, then there is no other edge of type* homeAg *with the same source vertex.*

Basis: The property is verified for the initial graph: all functions of the stated property are instantiated for $G0$. Since this involves only the base cases of functions definitions, the property is trivially evaluated to true.
Hypothesis ⇒ Inductive Step: Assuming that the property is valid for any reachable graph G, the proof requires 15 cases, depending on the considered rule. A TP would go through all these cases, that can be grouped into 3 classes:

Case Class 1: Edges of type homeAg *do not appear in the rule.* In this case the TP identifies, consulting the relations of LHS and RHS of the considered rule, that there are no deleted, created or preserved homeAg edges. Then, TP validates the property by using the induction hypothesis.

Case Class 2: There is an edge of type homeAg *preserved by the rule.* For proofs about preserved items we can use the following statement: first, there is an image using the match for all items that are in the LHS of the rule (rule applicability condition); second, items that are image of the LHS always satisfy the statements in G (by induction hypothesis). Consulting the relations of the relational structure, TP verifies that there is no other edges of type homeAg in the RHS of the rule. Since the edge of the RHS is preserved, the corresponding edge in G satisfies the property, and thus, there is no other edge of type homeAg with the same source vertex. For edges of type homeAg that are in the part that is not changed of the reachable graph, property is validated by hypothesis.

Case Class 3: There is an edge of type homeAg *created by the rule.* In this case, TP verifies the property for the RHS of the rule, but can not directly deduce if there are edges of type homeAg in vertices that are image of the match. This is an example of situation in which the user must use auxiliary properties to complete the proof. One may use the following property: if there is an edge with source in a vertex of type homeAck and target in a vertex of type MN, then there is no edge of type homeAg with source in this MN vertex. After this statement (and respective proof), the proof for this case can be completed.

Property 2 *For all edges of type* default*, its target vertex has an attribute of type* addr *with the same value of as the first component of the attribute of type* primaryCoA *of its source vertex.*

Basis: Analogous to proof of Property 1.
Hypothesis ⇒ Inductive Step: Here we also have 3 classes of cases:

Case Class 1: Edges of type default *do not appear in the rule.* Similar to the case 1 of previous analysis.

Case Class 2: Edges default*,* addr *and* primaryCoA *are preserved.* According to previous stated theorems, since the edges are preserved items they satisfy the property in G and thus, also in the resulting graph. For the preserved part of the reachable graph, the property is validated by hypothesis.

Case 3: There is an edge of type default *created by the rule.* The property is verified by consulting the relations that define the RHS of the rule. For the preserved part of the reachable graph, property is validated by hypothesis.

5 Final Remarks

In this paper we have described the two main steps, stating and proving of properties, in the process of verifying a mobile support for Internet protocol using attributed graph grammars. More specifically, we have adopted a relational and logical approach of attributed graph grammars, which allowed the use of mathematical induction as verification technique. The main advantage of such choice consisted on the perspective of using a theorem prover in the system analysis and on the possibility of verifying the system without using any kind of approximation for the model, even though it comprising an infinite state-space. Due to space limitations, we have restricted our analysis to only two properties. Many other interesting properties could also have been considered.

We plan to use event-B [11] and its theorem provers to implement the relational and logical approach of graph grammars. Moreover, we intend to extend the proposal to graph grammars with negative applications conditions [12]. These conditions restrict the application of a rule by asserting that a specific structure must not be present before or after applying the rule to a certain state-graph. This extension may raise our flexibility in the use of the relational approach for the specification of systems in all kinds of application areas.

References

1. Dotti, F.L., Ribeiro, L.: Specification of mobile code systems using graph grammars. In: FMOODS 2000. Kluwer (2000) 45–64
2. Ribeiro, L., Dotti, F., Bardohl, R.: A formal framework for the development of concurrent object-based systems. In: Formal Methods in Software and Systems Modeling. Volume 3393 of LNCS., Springer (2005) 385–401
3. König, B., Kozioura, V.: Towards the verification of attributed graph transformation systems. In: ICGT 2008. Volume 5214 of LNCS., Springer (2008) 305–320
4. Löwe, M., Korff, M., Wagner, A.: An algebraic framework for the transformation of attributed graphs. (1993) 185–199
5. da Costa, S.A., Ribeiro, L.: Formal verification of graph grammars using mathematical induction. In: Proc. of the Brazilian Symposium on Formal methods – SBMF2008. (2008) To appear in Electronic Notes in Theoretical Computer Science.
6. da Costa, S.A., Ribeiro, L.: Relational and logical approach to graph grammars. Technical Report 359, Porto Alegre: Instituto de Informática/UFRGS (2009)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2000)
8. Baldan, P., König, B.: Approximating the behaviour of graph transformation systems. In: Proceedings of ICGT '02 (International Conference on Graph Transformation). Volume 2505 of LNCS., Springer (2002) 14–29
9. Rozenberg, G., ed.: Handbook of graph grammars and computing by graph transformation: volume I. foundations. World Scientific Publishing Co., Inc. (1997)
10. Johnson, D., Perkins, C., Arkko, J.: Mobility Support in IPv6. RFC 3775 (Proposed Standard) (June 2004)
11. Hallerstede, S.: On the purpose of event-b proof obligations. In: ABZ 2008, Springer-Verlag (2008) 125–138
12. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundamenta Informaticae **26**(3-4) (1996) 287–313