DECIDABILITY AND TRACTABILITY OF PROBLEMS IN OBJECT-BASED GRAPH GRAMMARS *

Aline Brum Loreto

Leila Ribeiro

Laira Vieira Toscani

Instituto de Informática - PGCC

Universidade Federal do Rio Grande do Sul

{loreto,leila,laira}@inf.ufrgs.br

Abstract Object-based programming languages are being widely used to construct concurrent and distributed systems. In such systems, the complexity of performing some task is usually measured in terms of messages that are exchanged to perform the task, because communication is almost always the most time consuming operation in these environments. In this paper we analyze the problem of verifying if a message can be delivered. We model object-based systems as a graph grammars, and analyze this property as a graph grammar property. It turns out that this problem is NP-Hard. With suitable restrictions on the kind of graph grammar rules that are used, decidability can also be proven.

Keywords: object-based systems, graph grammars, complexity analysis

Introduction

Graph grammars have originated from the concept of formal grammars on strings by substituting strings by graphs [5]. Methods, techniques, and results for graph grammars have been studied since then, and applied in a variety of fields in computer science such as formal language theory, pattern recognition, software engineering, concurrent and distributed system modelling, database design and theory, etc. (see e.g., [4]). In particular, graph grammars are very well suited to the specification of concurrent and distributed systems: a (distributed) state of the system can naturally be represented by a graph and rules (where the left- and right-hand sides are graphs) describe possible state changes. The behavior of the system is then described via applications of these

*This work was partially supported by the projects PLATUS/ForMOS (CNPq and Fapergs), IQ-Mobile (CNPq and CNR) and GRAPHIT (CNPq and IB-DLR).

rules to graphs describing the actual states of a system. Rules operate locally on the state-graph, and therefore it is possible that many rules are applied at the same time. Graph grammars are appealing as a specification formalism because they are formal, they are based on simple but powerful concepts to describe behavior, and, at the same time, they have a nice graphical layout that helps non-theoreticians understand a graph grammar specification.

Practical applications involving concurrency and distribution aspects are hard to develop and reason about. Therefore, a specification formalism for this kind of application should offer verification methods (if possible, automated). For this, there must be a way to describe the desired properties of the system, and a way to prove them using the semantical models of the specification formalism. As graph grammars are an extension of string grammars (strings can be modeled by special graphs), we can try to generalize results from the theory of formal languages to graph grammars. However, as the main aim of a formal language is to describe a language and the aim of a graph grammar, as we are using them here, is to describe the behavior of a system, the kind of properties that are of interest are quite different. Graph grammars can also be seen as a generalization of Petri nets [11]. Therefore, we will also discuss in the conclusion the difficulties involved in the generalization of properties of Petri nets into properties of graph grammars, showing that read access to items, although very useful to let many actions occur in parallel, provides a source of verification problems.

As formal grammars on strings and Petri nets are special cases of graph grammars, all undecidability results of these areas also hold for graph grammars. Here we consider some restrictions for graph grammars, still allowing read access to items. This restricted class of graph grammars is called object-based graph grammars [3]. The restrictions imposed in this model implement the usual requirements of an object-based model, like encapsulation and communication via message passing. Object-based specification/programming languages seem to be a natural choice to describe reactive systems: entities that compose the application are modeled as *objects* and the reactions are triggered by *messages* and implemented as *methods*.

Two measures are of great interest for concurrent object-based systems: the number of messages that must be exchanged to complete some task (this gives us a space bound to perform a task); and the length of a round, that is, the length of the biggest causally related chain of messages needed to complete a task (all messages in such chain can not be delivered in parallel, and therefore this gives us a kind of time bound to perform a task). For these two measures, one must define what *completion of a task* means. In this paper we investigate the possible answer to this question, based on a rule application problem (RAGG), and show results about decidability and complexity of this problem.

The paper is structured as follows: Sect. 1 brings an informal introduction to object-based graph grammars; in Sect. 2 we introduce the problem RAGG; Sect. 3 shows results about decidability and complexity of this problem; and Sect. 4 compares it to other related problems and concludes our work.

1. Object-Based Graph Grammars

In this paper we consider an object-based system as being a system consisting of autonomous entities called *objects* that communicate and cooperate with each other through *messages*. Objects may have an internal state and relate to other objects within the system. The behavior of an object is described through its *reactions* to the receipt of messages (triggers). This reaction may be to change the object's internal state and/or send messages to other objects. An object may perform many (re)actions in parallel. A way to model object-based systems using graph grammars have been presented [3] inspired in the actor model [1]. The basic idea was to use graph grammars as a (graphical) language to specify a kind of actor systems.

To define object-based graph grammars, we have to identify within a graph grammar what are the *objects*, *messages* and *attributes*, and then show how to specify *methods* within this formalism. The structural part will be modeled by distinguishing different kinds of vertices and edges within the graphs that model states of the system (see Figure 1 (a)). There are many ways to define typing mechanisms for graphs, here we will use the concept of a typed graph[2, 10]. The idea of a typed graph is to use a graph, called *type graph*, to define the possible kinds of vertices and edges of a system, and an actual graph is then a graph consisting of instances of elements of the type graph. A typed graph can be described by a graph homomorphism relating each instance with its type. Besides distinguishing types of vertices and edges, for practical applications we usually use values belonging to carrier sets of algebras. Graphs with such values associated to vertices and/or edges are called *attributed graphs* [13, 6].

In our specification formalism, objects and messages will be modeled as vertices. A message must have as destination an object and may have as parameters objects and/or values belonging to data type sets. The internal state of an object consists of attributes, that may be references to objects and/or values. This graph (Figure 1 (a)) can be considered as a type graph for an object-based system, and therefore we will call it *object-based type graph*¹. Note that a type graph models kinds of objects and links that may be present in an actual state of the system, but say nothing about the number of elements of each kind that must be present at a particular state.

Figure 1. (a) Object-Based Type Graph OG (b) Object-Based Graph Grammar

¹Formally, the carrier set *Value* will be used to label the edges of a graph, that is, the edges *par_v* and *atr_v* are actually loops on vertices *Msg* and *Obj*, respectively.

For each specific object-based system we may have various types of objects and messages that are relevant for that application. Thus, to build a specification for an object-based system using graph grammars one must first define what we call the *application type-graph*. This graph must be typed over the object-based model type-graph. The resulting structure of a object-based graph grammar is illustrated in Figure 1 (a) ².

Object-based Application Graph. A typed graph G over a type graph T is a total graph morphism $type : G \rightarrow T$ (this morphism has three components, one function to map vertices, one to map edges and one algebra homomorphism to map the algebra of the instance graph to the one of the type graph). An object-based application graph is a graph typed over the object-based type graph where messages have only one destination.

Object-based Graph. An object-based graph G is a graph typed over an object-based application type graph where: *i*) Each message has all defined parameters; *ii*) An object does not have two times the same attribute.

Rules specify the behavior of the system in terms of local state changes. The left-hand side of the rule specifies a pattern that must be present in some state for the rule to be applied; the right-hand side shows the effect of the application of the rule; and the mapping from left- to right-hand side describes deletion (items that are not mapped), creation (items that are not in the range of the mapping) and preservation (items that are mapped). For an object-based graph grammar we will only allow rules that consume an element of type message, i.e., each rule represents a reaction to the kind of message that was consumed. Moreover, only one message may be consumed at a time by each rule. Note that the system may have many rules that specify reactions to the same kind of message (non-determinism), and that many rules may be applied in parallel if their triggers (messages) are present at an actual state. Many messages may be generated in reaction to one message. To make sure that a rule may be applied whenever its trigger is found in the actual state graph we will require that whenever a message appears in a graph, it has exactly all specified arguments and one destination.

Given object-based graphs $G1 = (GI1, type1, T)$ and $G2 = (GI2, type2, T)$, a morphism $g : G1 \rightarrow G2$ is a graph homomorphism $g : GI1 \rightarrow GI2$ that preserves types, that is, $type2 \circ g = type1$. This graph homomorphism may be partial on vertices and edges, but must be total on the algebra component. The compatibility condition required by the partial graph homomorphism is that the graph structure is preserved for the items in the domain of definition of the homomorphism, that is, preserved itens can not change types.

Object-based Rule. A morphism $r : L \rightarrow R$ between object-based graphs is a **(object-based) rule** iff L and R are finite r is injective, there is exactly one

²Formally, this structure can be defined as a doubly-typed graph grammar (see [16], [3] for the formal definitions). One of the advantages of defining explicitly the model type-graph within the specification is to ease the comparison among specifications with respect to different model graphs (once we relate the model graphs, the relationships among the specifications can be obtained automatically).

message vertex m in L , and this vertex is deleted by the rule. In this case, m is called **trigger** of r . Moreover, all attributes appearing in L must also appear in R (maybe with different values).

Object-based Graph Grammar. An **object-based graph grammar** is a tuple $GG = (AG, GI, R)$ where AG , the **type** of the grammar, is a finite object-based graph, GI is a finite graph typed over AG , called the **initial graph** of the grammar, and **Rules** is a finite set of object-based rules typed over AG .

The behavior of a graph grammar is based on the notion of derivation step, that is, rule application. A rule is applicable if all the items in its left-hand side are found in the graph representing the current state. The application of the rule performs the deletions and creations according to what is specified in the rule. Here we follow the (Single-Pushout) Algebraic Approach to Graph Grammars [15, 6]. The graph representing the state after the application of a rule $r : L \rightarrow R$ can be obtained as follows:

Occurrence. Given a rule $r : L \rightarrow R$ and a graph G , an occurrence for r in G is a total (typed) graph morphism.

Rule application. The graph representing the state after the application of a rule $r : L \rightarrow R$ can be obtained as follows: i) insert in the graph representing the state all items that are in R and not in L ; ii) Remove from the resulting graph, the items that are in L and not in R , and also all dangling edges, such that the result is a graph. This construction can be formally defined as the pushout in the category of (typed) graphs and partial graph morphisms [6].

The semantics of a graph grammar can be defined as the class of all computations that can be performed using the rules of the grammar starting with the initial state. These computations may be sequential or concurrent, giving raise to sequential and concurrent semantic models. As here we do not have to reason about parallelism, we will stick to this sequential model (where parallelism is described by interleaving). Note that, if we would have chosen a true concurrency semantical model, like unfolding [16] or concurrent derivations [10], the set of reachable graphs would be the same, as well as the set of rules that may be applied in each state. What changes is that true concurrency models allow more derivations to occur in parallel (interleaving is not equivalent to true concurrency in graph grammars [12] due to the hability to preserve items).

2. Definition of the Problem RAGG

To define when a task has been completed in a system we can use the generation of a message (end-message), or the fact that some procedure has been called. In graph grammars, these two would correspond to the application of a rule: the rule that generates a message, or a rule that starts some procedure. Thus, we can use the knowledge about the fact that a rule has been applied to describe when a task has been completed. In order to determine if a rule can be applied, we must verify if it is possible to generate the message that enables this rule, and if the attributes used by rule may reach the necessary values. The problem RAGG is the problem of application of a rule in graph grammars, to define it we use the notation of Garey and Johnson [9].

Definition 2.1 *RAGG Problem*

Instance: Object-based Graph Grammar $GG = (T, GI, R)$, and a rule $r \in R$.

Question: Is there a sequential derivation of GG in which the rule r is applied?

Definition 2.2 *RAGG- m Problem*

Instance: Object-based Graph Grammar $GG = (T, GI, R)$, and a rule $r \in R$.

Question: Is there a sequential derivation of GG in which the rule r is applied m times?

These two problems will be studied in many contexts, with graph grammars with different restrictions. Let us define some restrictions:

- 1) The rules have no attributes;
- 2) In left-hand side of a rule there exists exactly one message (restriction of object-based system);
- 3) The attributes values vary over a finite set;
- 4) The attributes values vary over the $\{T, F\}$ set;
- 5) Each attribute can change its status at most one time in the derivation;
- 6) Each rule can change the status of at most one attribute;
- 7) The status of an attribute can be changed only by one rule.

Graph grammars with restriction 1 were studied in [14], where it was shown that RAGG- m is decidable (considering restrictions 1 and 2). The cost was calculated (number of necessary rule applications to reach the m executions of r). The algorithm that calculates the cost has exponential complexity, however it was not proven that there is no polynomial algorithm nor that the problem is NP-Hard or NP-Complete. The same problem but with restrictions 2, 4, 5, 6 and 7 will be analyzed in the section 3 (Theorem 3). Restriction 2 is typical for object-based system: each action is triggered by one message. Restrictions 3 and 4 are finiteness assumptions (actually we can encode grammars having restriction 3 into grammars having restriction 4). Restriction 5 will be used to prove termination. Although it seems to be very restrictive for practical applications, in the conclusion we discuss a way to weaken this condition. Restrictions 6 and 7 are only used to allow easier proofs, they actually do not restrict the expression power of the grammar. Any grammar can be translated into a grammar satisfying 6 and 7.

3. Analysis of RAGG- m and RAGG Problems

In this section the complexity of the problems RAGG and RAGG- m will be investigated. We will reduce the satisfiability problem (SAT) into RAGG- m . The resulting grammar has many of the restrictions defined in Sect. 2.

Definition 3.1 *Reduction $SAT \propto RAGG-m$.*³

3

A problem L is NP-Hard if and only if Satisfiability reduces to L [9], that is, there is a deterministic polynomial time algorithm that transforms each instance of SAT into an instance of L , preserving the answer. This transformation (reduction) is denoted by $\propto$.

Let $I_{SAT} = (U, C)$, with variable set $U = \{u_1, u_2, \dots, u_n\}$ and clause set $C = \{c_1, c_2, \dots, c_m\}$, be an instance of the SAT problem. The instance I_{SAT} reduces to the instance $I_{RAGG-m} = (G, k, r)$ of RAGG, where:

- $G = (T, GI, R)$ is a graph grammar having T as type graph, GI as initial graph (Figure 2 (a) and (b), respectively), where $Bool = \{F, T\}$, and the rules set R is the union of the following rule sets:

Group AV: To each variable $u_i, i = 1, \dots, n$ we define two rules u_i^+ and u_i^- , representing the two possible value attributions to this variable (true or false). The scheme of rules of this group are depicted in Figure 3.

Group SV: Is composed by the rule described in Figure 4 (a), named r .

Group CVV: To each variable u_i present in clause c_j there is a rule named $c_j u_i^+$ as shown in Figure 4 (b). Analogously, to each variable u_i present in clause c_j there is a rule named $c_j u_i^-$ (Figure 4 (c)).

- m is the number of clauses in I_{SAT}
- r is the rule shown in Figure 4 (a)

Lemma 3.1 Let $GG = (T, GI, R)$ be as defined in Def. 3.1. For each $i = 1..n$, exactly one of the following sets of messages will be generated: $\{c_1 : u_i^+, \dots, c_m : u_i^+\}$ or $\{c_1 : u_i^-, \dots, c_m : u_i^-\}$ (representing the attribution of T/F to variable u_i , respectively). Each of these sets contains m messages (one for each clause in C). It means that each variable will have the same attribution in all clauses.

Proof. To each variable u_i , either rule u_i^+ or rule u_i^- can be applied (because both delete the message $u_i?$) generating for each clause one message $c_j u_i^+$ or $c_j u_i^-$, respectively.

Lemma 3.2 To each $j \in \{1, \dots, m\}$, at most one rule from $\{c_j u_1^+, c_j u_1^-, \dots, c_j u_n^+, c_j u_n^-\}$ can be applied, and therefore at most m SAT messages will be generated.

Proof. If rule $c_j u_i^+$ is applied to generate a message SAT, the value of the corresponding attribute $c_j Set$ must have been false and set to true. There is no rule that changes the status of $c_j Set$ to true again.

Now, we can prove that, when we have an affirmative answer to instance SAT, we have an affirmative answer to correspondent instance of RAGG- m .

Theorem 1 There is a Yes answer to the I_{SAT} of SAT iff there is a Yes answer to the I_{RAGG-m} instance of the RAGG- m problem.

Proof. ($\Rightarrow$) Suppose $I_{SAT} = (U, C)$ gives a Yes to SAT problem, i.e. there is a truth value attribution atr to the variables of U that makes all clauses in C true. We must find a sequential derivation σ of GG in which the rule r is executed m times. This derivation will be composed by $n + m + m$ derivation steps, as described below:

 Figure 2. (a) Type Graph T (b) Initial Graph GI

Figure 3. Rules Group AV

 Figure 4. (a) SV: Rule r (b) CVV: Rule $c_j u_i^+$ (c) CVV: Rule $c_j u_i^-$

- 1.. n steps: If atr assigns true to u_i , then consider the derivation where the rule u_i^+ is applied. This will have the effect that messages $\{c_1 : u_i^+, \dots, c_m : u_i^+\}$ will be generated and the trigger $\boxed{u_i^+}$ will be deleted (Lemma 3.1). Analogously, if atr assigns false to u_i , messages $\{c_1 : u_i^-, \dots, c_m : u_i^-\}$ will be generated.
- $n + 1$.. $n + m$ steps: For each clause $c_j \in C$, at least one of its components is true (because atr satisfies all clauses). Let u_i be that component. In this case, rule $c_j u_i^+$ can be applied because u_i^+ must have been generated in step i , generating one SAT message for clause j . Moreover, Lemma 3.2 assures that no other SAT message will ever be generated for this clause. The same would happen if $\neg u_i$ is the component that makes the clause true. Therefore, as the attribution atr makes all clauses true, we would generate m SAT messages.
- $n + m + 1$.. $n + m + m$ steps: As we have m SAT messages, rule r can be applied m times.

($\Leftarrow$) Let σ be a sequential derivation with m occurrences of rule r . For this to happen, m messages SAT must have been generated (because each application of r consumes one message SAT). That means that m messages of set CVV must have been applied before (because these are the only rules that generate SAT messages). Lemma 3.2 assures that, for all j , at most one rule of $\{c_j u_1^+, c_j u_1^-, \dots, c_j u_n^+, c_j u_n^-\}$ can be applied. This implies that each of the m SAT messages must have been generated for a different c_j . Moreover, for each u_i , either $\boxed{c_j : u_i^+}$ or $\boxed{c_j : u_i^-}$ is generated (Lemma 3.1). Consider the following attribution atr : if $c_j u_i^+$ was applied in σ then $atr(u_i) = \text{true}$; if $c_j u_i^-$ was applied in σ then $atr(u_i) = \text{false}$, for all i . If this attribution is well-defined, it would make all m clauses c_j true because if u_i is in c_j and $atr(u_i) = \text{true}$, the clause is true. The same holds for $\neg u_i$. This attribution is well-defined because rules $c_j u_i^+$ and $c_j u_i^-$ are mutually exclusive. Moreover, if $c_j u_i^+$ happens, there can be no other rule $c_l u_i^-$ that have been applied (this would mean that both $\boxed{u_i^+}$ and $\boxed{u_i^-}$ have been applied, and this is not possible – Lemma 3.1).

Theorem 2 *The reduction α can be done in polynomial time.*

Proof. The initial graph GI has n messages and m attributes, that is $n + 1$ vertices and $n + m$ edges and the carrier set of the attribute algebra has two elements (T and F), and therefore it can be constructed in polynomial time steps. The total number of rules is: in group AV $2n$; in group CVV $m * n$ rules, at most; in group SV, only one rule. Each rule in AV has exactly $m + 3$ vertices and $m + 1$ edges. Each rule in CVV has 4 vertices and 4 edges. And the r rule has 3 vertices and one edge. Thus, I_{RAGG} can be constructed in polynomial time.

Theorem 3 (NP-Hardness) *The problem RAGG- m with restrictions 2, 4, 5, 6 and 7 is NP-Hard.*

Proof. Follows directly from theorems 1 and 2.

Corollary 3.1 *Changing only the rule SV in definition 3.1 adding on the left-hand side the attributes $c_1 \text{Set}, \dots, c_m \text{Set}$, all set to T , we prove that RAGG is NP-Hard (with the same restrictions of the theorem).*

Theorem 4 *The RAGG problem with restrictions 2 and 3 is NP-Hard.*

Proof. It is enough to adapt the definition 3.1 placing the counter with value m in the initial graph, decreasing one in each rule $c_j u_i^+$ and $c_j u_i^-$ and condition your application by $m \neq 0$ and changed the r rule, placing the attribute *cont* with value zero in its right-hand side.

To prove that this problem is decidable we will define an algorithm that the builds a tree of all possible derivations that generate the necessary conditions for the execution of r . These derivations will first set the necessary attribute values, and then generate the trigger message (without changing the attribute set by the first step). To describe this algorithm, we will divide the rules of a grammar into classes, according to the attributes they change.

Definition 3.2 *Given a grammar $GG = (T, GI, R)$ and $r \in R$. We can partition the rules of GG in four classes: A, B, C and D . A is the class of rules that modify attributes present in left-hand side of r to values that disable the application of r and T_A is the set of these attributes. B is the class of rules that modify attributes present in left-hand side of r to values that enable the application of r and T_B contains this attributes. C is the class of rules that modify the attributes not present in left-hand side of r and T_C contains this attributes. And D is the class of rules that does not modify attributes and T_D is empty.*

This means that, in order to apply rule r , the rules of A can not be applied, the rules of B must be applied and the rules of C and D can or not be applied. Let G_{-A} be denote the GG grammar without the rules of A class.

Definition 3.3 *Causal sequence: Let $s = \langle r_1, r_2, \dots \rangle$ be a sequence of rules in G_{-A} , and t_i the attribute changed by r_i rule, if r_i modify some attribute. Then s is a Causal sequence if to each r_i , $\langle r_1, r_2, \dots, r_i \rangle$ is a sequence of application of rules from GI that generates the conditions to r_i to be applied.*

Now, consider the G_{-A-M} the grammar equal to G_{-A} , but without messages. The proofs of lemmas 3.3, 3.4 and 3.5 are straightforward and were omitted.

Lemma 3.3 *If there is a causal sequence $\langle r_1, r_2, \dots, r_k \rangle$ from rules of $B \cup C$ containing all the rules of B and possible some of C then r can be applied in the G_{-A-M} grammar.*

Lemma 3.4 *If r is not applicable in G_{-A} then r is not applicable in GG .*

Lemma 3.5 *If there are derivations $T1 : GI \rightarrow GF$ and $T2 : GF \rightarrow GM$, where GF satisfies all the attributes present in left-hand side of r and $T2$ change no attributes necessary to execution of r and does not consume the necessary messages for r , then r is applicable in GM .*

Definition 3.4 *Consider all causal sequences $\langle r_1, \dots, r_k \rangle$ satisfying Lemma 3.3 with the same subsequence of rules of B . From these sequences and the*

initial graph GI , the tree illustrated in Figure 5 is built. Many derivations can be constructed based on these sequences, the upper indices identify these variations. Each leaf vertex is the graph resulting from the application of the $d^j = d_1^j; \dots; d_k^j$ derivation that precedes it, where each d_i^j is a derivation that contains no rule that modifies attributes and where the last applied rule is r_i .

Figure 5. Tree of Derivation Sequence

Theorem 5 *The RAGG problem with restrictions 2, 4, 5, 6 and 7 is decidable.*

Proof. Let $GG = (T, GI, R)$ be a graph grammar with the restrictions 2, 4, 5, 6 and 7 and $r \in R$. Let A, B, C, D be defined from GG and r , by definition 3.2. In case of $C = \emptyset$, consider the follow algorithm: For each sequence $\langle r_1, \dots, r_k \rangle$ that satisfies Lemma 3.3 do the steps i) and ii) below. If this sequence does no exist, r is not applicable in GG (Lemma 3.4). This problem is obviously decidable because the sequence set to test is finite.

i) Build the derivation tree of Figure 5. Each d_i is a derivation is a gramar without attributes and therefore we can calculate [14] if r_i is applicable and which (finite) derivation creates the needed conditions to apply r_i ;
 ii) Verify, for each leaf of the tree, if r is applicable in the grammar (T, GI', R') , where GI' is the leaf graph removing the attributes; R' is the rule set D of GG . For a grammar without attributes, the problem is decidable [14].

Using Lemma 3.5, under the condition of $C = \emptyset$, this algorithm decides if r is or not applicable in GG . As the number of sequences $\langle r_1, \dots, r_k \rangle$ satisfying Lemma 3.3 is finite the problem is decidable.

The condition $C = \emptyset$ corresponds to the case where rule r reads all the attributes. When this condition is not true we must replaced r by rules that read all the attributes, considering all the possible assignments to the attributes not read by r . If one of these rules is applicable in the new graph grammar, r will be applicable in GG . As the number of these rules is finite, it is possible to decide whether r is applicable in GG or not.

4. Conclusion

This paper investigates the complexity of the problems *RAGG* and *RAGG-m* in object-based graph grammars with some restrictions. The problems stayed in different complexities classes depending on the considered restrictions. For graph grammars with restrictions 2, 4, 5, 6 and 7, the problem *RAGG* was proven decidable (theorem 5) and NP-Hard (theorem 3.1). Adapting the reduction of definition 3.1 we can prove that, with restrictions 2 and 3, the *RAGG* problem is also NP-Hard.

Graph Grammars are a generalization of Petri Nets. The complexity of verification of properties have been exhaustive investigated for Petri Nets. One of the most investigated ones is the reachability problem. This problem is in different classes of complexities depending on the characteristics of the net. Reachability in Conflict-free Petri Nets is in P, for Petri Nets without cycles it is in NP-Complete class and for some extended Petri Nets like with inhibitor arcs the problem is undecidable [7, 8]. It is to expect that for grammar graphs the same occurs, the corresponding problem will be in different classes depending on the particularities of the investigated grammar.

In Formal Languages, given a grammar G , the properties usually investigated are “does $w \in L(G)$?” or “ $L(G) \neq \emptyset$?”. The former corresponds to the reachability problem that was already discussed above. The latter is not very useful because it would mean to verify if a system can evolve (at least one step). When using a graph grammar to represent a concurrent system a more interest property is to know if a r rule can be applied. This property in Context Free Grammar (*CFG*) corresponds to find out if a non-terminal is reachable. This test is usually done in the phase of simplification of grammar, the algorithm is simple and of polynomial degree. A graph grammar without attributes in which rules have an unique message in their left-hand sides is like the *CFG*. The messages of the type graph are the variables (non terminals) and the messages in the initial graph are the initial symbols. It is therefore to hope that the problem of verifying if a rule is applicable is a problem in P (has algorithm of polynomial order) for this particular case of graph grammar. Yet, the problem to know whether a rule can be applied m times does not have easy solution, even in *CFG*.

Although this paper has theoretical contributions, for the practice we still have to find conditions that are not too restrictive for system specification. The ideal solution would be that these restrictions should be enough to put the problem in a more promising complexity class, like P, allowing for efficient verification. On the other hand, the restrictions presented here could be weakened to detect the limits of decidability of this problem. Concerning this topic and disregarding restriction 1 (that is too restrictive for practical applications and was not imposed in the results presented in this paper) and 2 (that is usually required for practical applications), we can see that the restrictions that really matter for the complexity and decidability results are restrictions 3 and 5, that state that the attribute sets are finite and attributes may change only once in a derivation, respectively. Finiteness is a typical requirement when thinking of

verification problems. Restriction 5 was used to assure the termination of the process. It could probably be weakened to imposing the existence of a well-founded partial order on the sets of attributes that change during a derivation, and requiring that each rule that changes such attributes sets them to values closer to the bottom (or top) of this partial order. In this sense, the application of the last rule (r) would mean the end of the execution of the desired task (this rule could change attributes disregarding the partial order, to put the system in some initial state again).

References

- [1] Agha, G. *Actors: a model for concurrent computation in distributed systems*, MIT Press, 1986.
- [2] Corradini, A., Montanari, U. and Rossi, F. *Graph processes*, *Fundamentae Informatica*, vol. 26, no. 3-4, 1996, pp. 241-265.
- [3] Dotti, F. and Ribeiro, L., *Specification of mobile code systems using graph grammars*. In S. Smith and C. Talcott, editors, *Formal Methods for Open Object-based Systems IV*, pages 45-64. Kluwer Academic Publishers, 2000.
- [4] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*, World Scientific, 1999.
- [5] H. Ehrig, M. Pfender and H. Schneider, *Graph grammars: an algebraic approach*, Proc. IEEE Conference SWAT'73, 1973, pp. 167-180.
- [6] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner and A. Corradini, *Algebraic approaches to graph transformation II: Single pushout approach and comparison with double pushout approach*, in *The Handbook of Graph Grammars, vol. 1: Foundations*, World Scientific, 1997, pp. 247-312.
- [7] J. Esparza and M. Nielsen. *Decidability Issues for Petri Nets-a survey*, J. Inform. Process. Cybernet. EIK 30 (1994) 3, 143-160.
- [8] J. Esparza, *Decidability and Complexity of Petri Net Problems-An Introduction*, Lecture Notes in Computer Science 1491, Springer, 1998, pp. 376-428.
- [9] M. R. Garey & D. S. Johnson, *Computers and Intractability: a guide to the theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.
- [10] M. Korff, *Generalized graph structures with application to concurrent object-oriented systems*, Ph.D. thesis, Technical University of Berlin, 1995.
- [11] Korff, M. and Ribeiro, L., *Formal relationships between graph grammars and Petri nets*, Lecture Notes in Computer Science 1073, Springer, 1996, pp. 288-303.
- [12] M. Korff and L. Ribeiro, *True concurrency = interleaving + weak conflict*, *Electronic Notes in Theoretical Computer Science*, vol. 14, 1998.
- [13] M. Löwe, M. Korff and A. Wagner, *An algebraic framework for the transformation of attributed graphs*, *Term Graph Rewriting: Theory and Practice*, John Wiley & Sons, 1993, pp. 185-199.
- [14] A. Loreto, L. Ribeiro and L. Toscani, *Complexity Analysis of Reactive Graph Grammars*, *Revista de Informatica Terica e Aplicada*, vol. VII(2000). Porto Alegre, Instituto de Informatica, UFRGS, pp.109-128.
- [15] M. Löwe, *Algebraic approach to single-pushout graph transformation*, *Theoretical computer Science*, vol. 109, 1993, 181-224.
- [16] L. Ribeiro, *Parallel composition and unfolding semantics of graph grammars*, Ph.D. Thesis. Technical University of Berlin, 1996.