Theoretical Computer Science 686 (2017) 25–77

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Theorem proving graph grammars with attributes and negative application conditions

Simone André da Costa Cavalheiro [b,*], Luciana Foss [b], Leila Ribeiro [a]

[a] Informatics Institute, Federal University of Rio Grande do Sul (UFRGS), Brazil
[b] Technology Development Center, Federal University of Pelotas (UFPel), Brazil

ARTICLE INFO

Article history:
Received 8 October 2015
Received in revised form 17 April 2017
Accepted 26 April 2017
Communicated by U. Montanari

Keywords:
Graph transformation
Theorem proving

ABSTRACT

Graph grammars may be used to formally describe computational systems, modeling the states as graphs and the possible state changes as rules (whose left- and right-hand sides are graphs). The behavior of the system is defined by the application of these rules to the state-graphs. From a practical point of view, the extension of rules to enable description of extra conditions that must be satisfied upon rule application is highly desirable. An example is the specification of *negative application conditions*, or NACs, that describe situations that prevent the application of a rule. This extension of the basic formalism enhances the expressiveness of rules, generally allowing simpler specifications. Another extension that is fundamental for practical applications is the possibility to use data types, like natural numbers, lists, etc., as attributes of graphical elements (vertices and edges). Attributed graph grammars are well-investigated and used. However, there is a lack of verification techniques for this kind of grammar mainly due to the fact that data types are typically infinite domains, and thus techniques like model checking can not be used directly (without abstraction constructions). The present work provides a theoretical foundation for theorem proving graph grammars with negative application conditions and attributes. This is achieved by generating an event-B model from a graph grammar. Event-B models are composed by sets and axioms to define types, and by states and events to describe behavior. After constructing the event-B model that is semantically equivalent to a graph grammar, properties about reachable states may be proven using the various theorem provers available for event-B in the Rodin platform. This strategy allows the verification of systems with infinite-state spaces without using any kind of approximation.

1. Introduction

Graph Grammars (or Graph Transformation Systems) [27] are well-suited for the formal specification of applications in which states have a complex topology (involving not only many types of elements but also different types of relations between them) and in which behavior is essentially data-driven, that is, events are triggered by particular configurations of the state. States are modeled by graphs and state changes, or events, by rules (that transform graphs). Many reactive systems are examples of this class of applications, like protocols for distributed and mobile systems, biological systems, and many others. Additionally to the complex states and reactive behavior, concurrency and non-determinism play an essential

* Corresponding author.
E-mail addresses: simone.costa@inf.ufpel.br (S.A.C. Cavalheiro), lfoss@inf.ufpel.edu.br (L. Foss), leila@inf.ufrgs.br (L. Ribeiro).

http://dx.doi.org/10.1016/j.tcs.2017.04.010
0304-3975/

role in this area of applications: many events may happen concurrently, if they all are enabled, and the choice of occurrence between conflicting events is non-deterministic. Originally, graph grammar rules specified only patterns that must be present to trigger rules, in [42] an extension was proposed allowing rules to specify also negative application conditions (short NACs), that are patterns that hinder the application of the rule. Such application conditions are commonly used in non-trivial specifications because the number of rules required to specify a system might be greatly reduced. The use of values defined by abstract data types [31] as attributes of vertices and edges complements the approach. Attributed graph grammars allow the use of (elements of) data types like natural numbers, lists, etc., as well as variables and terms in rules. Most practical applications using graph grammars are based on approaches with attributes and NACs.

To check whether a system specified using graph grammars has the desired properties, we may use verification techniques. There are various approaches available for graph grammars, some of them are reviewed in Section 2. With static analysis we may typically find problems related to typing, but also analyze whether the potential conflicts and dependencies among rules are the expected ones for the system being modeled (using critical pair analysis), or even check whether it is possible that specific sequences of rule applications generate desired results (by constructing concurrent rules). But certainly, correctness of behavior in a more general sense can not be ensured statically. Model checking techniques inspect computations of a system to verify whether the presented behavior, given by the application of the rules, corresponds to the expected one (described, for example, as formulas in some logic). The number of computations of a system depends on the number of different choices of behavior that are available at each state, and this, in turn, depends on the number of rules of the system but also on the number of different ways in which each rule may be applied in the state. Even with only one rule there may be a huge (or even infinite) number of choices if the state graph is large and/or the rule has a variable ranging over an infinite type domain. One of the strengths of graph grammars is the ability to model complex states (as graphs), possibly using attributes from general data types, and therefore it is to expect that even small specifications indeed induce a huge number of complex reachable graphs. This means that analysis techniques that involve building the whole (or even parts of) state-space of a system, like model checking, are of limited use for graph grammars with NACs and attributes. Since these grammars are the ones that are mostly used in practical applications, we started in [21,22] investigating the use of the theorem proving technique in this context.

In order to prove properties of a system using theorem proving, we must encode the system (states and behavior) and the properties to be proven in some logic, and then use deduction to verify that the system satisfies the properties. This encoding of the system, in our case represented by a graph grammar, is a critical task, since we must ensure that it preserves the semantics of the system. Moreover, the encoding must be also useful, in the sense that it enables to express and reason about the desired properties in a suitable way. For example, consider the property *"there is always exactly one node of a specific type in any state"*. This kind of property is at a level of abstraction in which we need to be able to inspect the components of the state (vertices and edges of a graph) to verify it. This means that the encoding of the system into the logic must be such that it allows us to easily reason about such components. We may be interested in many kind of properties, like properties of computations and/or properties of reachable states. Again, since the possibility to represent complex states (as graphs) is one of the strengths of the graph grammar formalism, proving properties about reachable states is especially important. In our previous work we introduced a relational approach to graph grammars, providing an encoding of graphs and rules into relations, that allowed to generate an event-B [4] model corresponding to a graph grammar. In event-B, models are composed of states, which are sets of variables, and events, that specify possible changes of values of state variables. Abstract data types may be defined as types for variables. Events may have complex guards. The encoding of graph grammars in event-B was claimed to be equivalent to the single-pushout approach to graph grammars [30], and was inspired by Courcelle's research about logic and graphs [18]. Properties about the reachable states of the system could then be stated as *invariants*, using First-Order Logic with Set Theory, and proven using theorem provers available for event-B specifications [20,70].

In this paper we completely formalize this translation, prove the preservation of behavior and extend the approach to deal with graph grammars with negative application conditions and attributes. In [22] we defined the theoretical foundations of the logical description of graph grammars with attributes, which is the basis for the work developed here in encoding these grammars as event-B models to allow formal verification. First we formally define the encodings for the Single- and Double-Pushout (SPO and DPO) graph grammar approaches including NACs, proving that the encoding preserves the semantics. We also show that, since the DPO does not allow rule applications that would have side effects, the encoding of DPO-rules can be simplified (because it is possible to determine which elements will be the deleted/created just by analyzing the rule). Then we provide an extension to consider (the translation of) attributes. The encodings of SPO and DPO with NACs and attributes are done at the set-theoretical level because this makes expressing and proving properties easier. Moreover, these encodings may be used as basis for the development of graph grammar tools since most tools actually work at the set-theoretic level to build the result of rule applications, and we prove the correctness of the proposed encodings with respect to the corresponding SPO and DPO categorical definitions.

The paper starts with a discussion about verification of graph grammars (Sect. 2), contextualizing and motivating the proposed work. Section 3 defines the mathematical notation used throughout the paper and briefly introduces the event-B formalism. Section 4 presents a definition of graph grammars with NACs together with a working example, as well as a set-theoretic construction for rule application. The translation of graph grammars with NACs to event-B is described in Sect. 5 and the extention of the approach to deal with attributes is presented in Sect. 6. Section 7 shows how the proposed approach can be used to verify properties of infinite-state graph grammars and Sect. 8 contains a discussion about the