Verifying object-based graph grammars

An assume-guarantee approach

Fernando Luís Dotti · Leila Ribeiro · Osmar Marchi dos Santos · Fábio Pasini

Abstract The development of concurrent and reactive systems is gaining importance since they are well-suited to modern computing platforms, such as the Internet. However, the development of correct concurrent and reactive systems is a non-trivial task. Object-based graph grammar (OBGG) is a visual formal language suitable for the specification of this class of systems. In previous work, a translation from OBGG to PROMELA (the input language of the SPIN model checker) was defined, enabling the verification of OBGG models using SPIN. In this paper we extend this approach in two different ways: (1) the approach for property specification is improved, enabling to prove properties not only about possible OBGG derivations, but also about the internal state of involved objects; (2) an approach is defined to interpret PROMELA traces as OBGG derivations, generating graphical counter-examples for properties that are not true for a given OBGG model. Another contribution of this paper is (3) the definition of a method for model checking partial systems (isolated objects or a set of objects) using an assume-guarantee approach. A gas station system modeled with OBGGs is used to illustrate the contributions.

Keywords Model checking, Partial systems, Graph grammars, Object-based systems, Reactive systems

1 Introduction

Concurrent and reactive systems are becoming very important. Almost all computers are networked, sharing resources and services both locally and globally; grid computing is being used to cope with a large number of problems nowadays. However, concurrent systems are inherently more complex to understand and reason about than sequential ones. In contrast to typical sequential systems, the execution of a concurrent system is usually non-deterministic due to many aspects, like scheduling policies and message delivery times.

Assuring the correctness of a concurrent system involves reasoning about all possible execution paths, or computations. Even in small concurrent systems, the number of computations may be enormously large, and a systematic way to analyze them becomes mandatory. One way to cope with this problem is through division of the system into manageable parts and application of
analysis techniques for each part separately. This strategy, besides being an approach to deal with the state-space explosion problem, is also well suited for various classes of systems. Thus, the construction and analysis of partial systems is becoming an important issue.

Open systems, i.e. systems that are supposed to communicate with a priori unknown other systems, are becoming very common and are inherently partial. Likewise, another important field is the distributed development of software, where each team is meant to work on part of a whole system. In such cases a clear definition of interfaces and a way to reason about each part separately are needed in order to rely on the development of each part and to ensure the correctness of the whole.

Formal methods provide a precise way to describe computational systems and reason about their behaviors. A large number of formal specification methods suitable for concurrent systems have been proposed in the literature, for example different kinds of Petri nets [40,42], process algebras like CCS [38], TLA [32], I/O automata [35], etc. However, in spite of the advantages of using formal methods in system development, they are not very used in industry. One of the reasons is that users find the mathematical notation difficult to understand. Here, we use a formal specification method called Object-based graph grammars (OBGGs) [19] that provides the users with a visual language that is based on a few but powerful concepts and follows the object paradigm, familiar to most users. The language itself is a restricted form of graph grammars (GGs) [44], with embedded notions used to model reactive object-based systems with asynchronous message passing as the underlying communication mechanism.

Due to their declarative characteristic, GGs are naturally well suited for concurrent systems specification. OBGGs preserve this advantage, allowing inter- as well as intra-object concurrency. OBGGs also capture the main abstractions to represent reactive and distributed systems considering the asynchronous computation model: (1) due to the encapsulation, objects can be located in different computational nodes, keeping their internal state; (2) communication takes place through messages, which are present in the specification method; and (3) since incoming messages are computed in a non-deterministic way, the delays for processing as well as message exchange are unbounded (which characterizes the asynchronous computation model). Therefore, OBGGs are well suited to specify reactive, concurrent and distributed systems.

Models specified using OBGGs can be analyzed using simulation [10,15] and verification (by model checking) [18]. Besides these analysis methods, there is also the possibility of generating code for execution in a real environment, via a mapping to the Java programming language [15]. A tool to assist the modeling and reasoning of OBGG systems has also been developed [16]. Various models have been defined and analyzed using OBGGs: mobile code applications [19], a pull-based failure detector [20], active networks [21], distributed election in a ring [17], dining philosophers [18], and readers and writers [45]. By using the methods and tools mentioned above, a framework to assist the development of concurrent and also distributed systems was defined. The innovative aspect of this framework is the use of OBGGs as the underlying unifying formalism. Moreover, in [20] we introduced the representation of classical fault models for distributed systems in OBGG models, allowing one to reason about a distributed system in presence of such faults.

Systems described with OBGGs can be verified using the model checker SPIN [30]. OBGG models can be translated to (semantically equivalent) PROMELA code (PROMELA is the input language of SPIN) and then the desired properties can be verified. One important aspect about this translation-based method for verification is that the properties (to be verified) are specified over OBGG models, instead of translated PROMELA codes, considering events of the OBGG model (applications of rules). However, counter-examples obtained from verifications using the SPIN model checker have no corresponding meaning to OBGG models, because they are generated over translated PROMELA codes. Moreover, until now, our approach supported only properties about events (names of rules) – it was not possible to prove properties using the states of the computation.

The first two contributions of this paper tackle these problems by extending the approach defined in [18] with: (i) the possibility to consider states of objects in property specifications; and (ii) an automatic way to generate graphical counter-examples, as OBGG derivations, for properties that do not hold for an OBGG model.

The object paradigm naturally suggests that the system is built of components (objects). At the specification level, this means that we should be able to specify and verify each component alone before building the whole system. Since components are typically reactive, an abstract description of the behavior of the environment in which they are meant to run is needed to simulate them and verify their properties. That is, we assume some behavior of the environment and guarantee that the component behaves correctly (provided that the environment behaves as assumed). The third contribution of this paper is (iii) the definition of a method for model checking partial systems (isolated objects or