Model checking RAISE applicative specifications

Juan I. Perna¹ and Chris George²

¹ Computer Science Department, The University of York, Heslington, York YO10 5DD, UK. E-mail: juan.perna@cs.york.ac.uk
² United Nations University International Institute for Software Technology, Macao SAR, China

Abstract. Ensuring the correctness of a given software component has become a crucial aspect in software engineering and model checking provides an almost fully automatic way of achieving this goal. Due to the scalability problems of the model checking technique, it has become popular to apply it at early stages in the development process, when the size of the model is much smaller than the final code. Properties proved in this way can be shown to hold at the implementation level provided that the final code refines the original specification. In this paper we focus on the main issues for adding model checking functionality to the RAISE specification language (RSL) and present the semantic foundations of our current approach for doing so. We also describe a way to use model checking to verify RAISE confidence conditions, ensuring the soundness and completeness of the results checked in this way. We then present the most interesting details of the implementation of a tool that follows the described approach. Finally, we illustrate the application of the technique with two case studies: a Digital Multiplexed Radio Telephone System and the Mondex electronic purse.

Keywords: RAISE, model checking, applicative specifications

1. Introduction

Model checking is a method to algorithmically verify formal systems [CGP99]. This is achieved by verifying if the model, often derived from a hardware or software design, satisfies a formal specification. The model is usually expressed as a transition system where a set of atomic propositions is associated with each node. The nodes represent states of a system, the transitions represent the actions that may alter the state, while the atomic propositions represent the basic properties that hold at each point of execution.

The utilization of model checking for software verification has been subject of significant research and study during the last decades [CGP99, BBF⁺98, FW06]. The increasing popularity of model checking is due to the high level of automation achieved (compared to other verification techniques such as testing or proof). Moreover, the ability to produce counterexamples when a given property is not satisfied provides useful insight when debugging.
a system. On the other hand, as the model checking technique is based on the calculation (and sometimes also representation) of all possible reachable states by the system under study, it suffers from the *state explosion* problem [BCM+90] (i.e. the size of the computation increases exponentially with respect to the size of the original problem). To overcome the state explosion problem, models often require considerable manipulation (see Sects. 5.1.1 and 5.2.2 for examples) and this makes model checking less useful for verification, because it is difficult to be certain that the manipulations correctly preserve behaviour. Similarly, reducing the size of the models might obscure problems that show up only in larger ones. Hence model checking is often seen as a technique for debugging rather than for verification, and this work supports that view. Formal verification is better handled by theorem provers, with model checking used to check systems before considerable effort is used in attempting to prove them.

*Formal methods* can be defined as mathematically based techniques for the specification, development and verification of software and hardware systems. The approach is especially relevant in high-integrity systems, for example where safety or security is important, to help ensure that errors are not introduced into the development process.

The Rigorous Approach to Industrial Software Engineering (RAISE) is a “formal specification language that aims at providing a sound notation … for capturing requirements and expressing the functionality of software”[Gro92]. In RAISE, software is developed in a step-wise manner starting from a high level description of the system and progressing by successively adding details towards a more concrete (and thereby closer to the implementation) specification. This style of development combines well with model checking because it allows the verification of properties in the early stages of the development process, when the more abstract specification is available to be model checked. Once verified, the RAISE development process [Gro95] warrants the preservation of the properties until the actual implementation of the system.

RAISE specification language (RSL) is a modular language, and its refinement relation is strong enough to allow substitution of a module by a refinement of it, so modules can be developed independently.

Regarding the existing verification mechanisms available for the RAISE language, our approach contributes mainly in two directions:

**Automation** RAISE provides several tools supporting verification, such as code generators to various languages in order to run the specification’s code [WG00, AG00], test case generators [DA03] (including test coverage analysis and mutation testing), and a translator to PVS [DG02] so important properties or invariants from the specification can be proved correct. Refinement of one module by another is also checked using PVS. Testing as a means of verification is, of course, limited. Theorem proving can be complete, but generally requires a great deal of involvement from the user.

**Confidence condition verification** Confidence conditions are conditions that should be true if a RAISE module is to be consistent. They include function preconditions and postconditions, and values being in subtypes. Confidence conditions can be checked either by hand or with the help of the translators (for example, by including run-time checks for them when generating C++ code). Regarding soundness and completeness, the strongest approach is provided by the translator to PVS. This tool encodes most of the confidence conditions as PVS “type check conditions” [Geo01] and the rest as theory lemmas. There remains, however, the problem of an inconsistent RAISE specification allowing the proof of invalid confidence conditions [Geo03]: any claim can be proved from an inconsistent specification.

Considering previous work on the field, two approaches to model checking are possible: to implement a brand new model checker tailored specifically to the language we are verifying [Fre05, Eur, CDH+00, BR02] or to use an already developed model checking tool and to translate from the source language into the model checker’s language [SW05, Hav99, TSG06]. Developing a brand new model checking engine for RAISE would have the advantages of eventual better efficiency (compared with other more general model checkers if some particularities from RAISE’s language can be used in order to achieve more efficient algorithms) and a closer “fit” of the model checker’s language with RSL (no additional syntactic or semantic restrictions apart from the ones inherent to the model checking technique). On the other hand, using an already developed model checker provides the security of a well tested verification engine combined with the possible advantages from future developments of the tool that can be immediately used if backward compatibility is preserved.