Assuring property conformance of code generators via model checking

Sven Jörges¹, Tiziana Margaria² and Bernhard Steffen¹

¹ Chair of Programming Systems, Technische Universität Dortmund, Dortmund, Germany. E-mail: sven.joerges@tu-dortmund.de
² Service and Software Engineering, Universität Potsdam, Potsdam, Germany

Abstract. Automatic code generation is an essential cornerstone of today’s model-driven approaches to software engineering. Thus a key requirement for the success of this technique is the reliability and correctness of code generators. This article describes how we employ standard model checking-based verification to check that code generator models developed within our code generation framework Genesys conform to (temporal) properties. Genesys is a graphical framework for the high-level construction of code generators on the basis of an extensible library of well-defined building blocks along the lines of the Extreme Model-Driven Development paradigm. We will illustrate our verification approach by examining complex constraints for code generators, which even span entire model hierarchies. We also show how this leads to a knowledge base of rules for code generators, which we constantly extend by e.g. combining constraints to bigger constraints, or by deriving common patterns from structurally similar constraints. In our experience, the development of code generators with Genesys boils down to re-instantiating patterns or slightly modifying the graphical process model, activities which are strongly supported by verification facilities presented in this article.

Keywords: Extreme Model-Driven Development, Code generation, Model checking, Verification

1. Introduction

Automatic code generation is a key feature of model-driven approaches to software engineering. It has several advantages such as the elimination of manual coding errors, and it provides a fast track to a deployable and testable system/application. Furthermore, it disburdens developers from writing boilerplate code, which is often a highly repetitive and cumbersome task, and by doing this, it shifts the attention back to the primary concern, the application-level logic. Of course, an indispensable requirement for the success of this approach is the reliability and correctness of the corresponding code generators.

In [JMS08], we presented Genesys, a framework for the high-level construction of code generators along the lines of the Extreme Model-Driven Development (XMDD, see Sect. 2) paradigm. In this framework, code generators are modeled on the basis of an extensible library of well-defined building blocks. We showed that constructing code generators this way offers several advantages, such as the high potential for reuse:
The building blocks, as well as whole parts of code generators covering features or aspects like error-handling or input/output-related tasks, can easily be reused for constructing new code generators. This enables short development cycles and a fast evolution of the generation library. We also showed that the approach perfectly integrates with other classical, well-established concepts, such as bootstrapping from the field of compiler construction.

In this article we focus on another advantage. The XMDD approach comprises model checking-based [CGP01, QS82] verification, which can be applied to the code generators: As the code generators are realized as formal models, they are amenable to such techniques. This enables us to check code generation rules or constraints, e.g. guaranteeing the complete processing of all input data or the correct order of generation steps. Genesys contains a large and steadily growing library of such constraints, which greatly improves the overall quality and reliability of the code generators.

In [JMS08], we already briefly motivated the use of model checking in Genesys with several simple example constraints. As the main contribution of this article, we want to elaborate on this by examining more complex constraints for code generators, which even span entire model hierarchies. We also show how we constantly extend our knowledge base for code generation by e.g. combining constraints to bigger constraints, or by deriving common patterns from structurally similar constraints. For our experiments, we used the model checker GEAR [BMRS07a, BMRS07b] to check whether the code generators conform to the constraints, which we specified graphically using the FormulaBuilder [JMS06, JMS08].

In our experience, the development of code generators with Genesys boils down to re-instantiating patterns or slightly modifying the graphical process model. These activities are strongly supported by the application of model checking presented in this article, as it helps leveraging the increasing body of domain knowledge during code generator construction.

In the following sections, we first will describe jABC, which is a basic framework that enables the development according to XMDD (Sect. 2). Afterwards, we present Genesys, which is based on the jABC (Sect. 3), and we briefly recapitulate the main ideas presented in [JMS08]. Section 4 outlines the verification facilities of the jABC, which we apply to check the property conformance of Genesys' code generators. Subsequently Sect. 5, the main section, describes and exemplifies the verification of code generators along an elaborate example. Finally, we discuss some related work (Sect. 6), before we conclude with Sect. 7.