Static Type Checking of Model Transformation Programs

Zoltán Ujhelyi

Model-driven development (MDD [1]) is becoming widely accepted in system and software engineering. MDD facilitates the systematic use of models from the very early phase of the design procedure: high-level, visual engineering models are used to capture system requirements and design, followed by the automatic generation of the source code and configuration files of the target application.

A key factor in the successful adoption of MDD is the use of model transformations utilized for various tasks, such as formal model analysis or code generation. However, model transformations in most cases are still written manually in industrial practice as a regular piece of software.

As more and more complex model transformations are developed, ensuring the correctness of the transformation programs becomes increasingly difficult. Nonetheless, detecting errors is required as they can propagate into the developed application, or invalidate the results of formal analysis.

Methods for ensuring correctness of computer programs such as *static analysis* are applicable for transformation programs as well. Static analysis represents a set of techniques for computing different properties of programs without their execution, used extensively in compilers for optimization and also for program verification.

In case of dynamically typed programming languages (such as Javascript) typing errors are one of the most common programming errors. They most often lead to misleading output rather than a runtime exception making them hard to trace. Static type checker tools address these problem by inferring the types of every program variable and validating all their use.

The current paper presents a static type checker component for early detection of typing errors in model transformation programs. The component was implemented for the VIATRA2 [2] model transformation framework (a general Eclipse-based modeling framework developed at BME-DMIS) based on graph transformations (GT) theory, a declarative, rule-based specification paradigm. Complex model transformations are defined by a combination of abstract state machines (ASM) and graph transformation rules. The ASM parts are dynamically typed that necessitates type checking, while GT rules are statically typed that provides information for efficient type inference.

Our approach describes type safety as constraint satisfaction problems: the type system is mapped to special integer sets; and the type information inferrable from the transformation programs are represented as constraints using these sets. Proper error messages are handled by dedicated back-annotation from the constraint domain. For performance considerations the rules and patterns of transformation programs are analyzed separately, and the partial results are described as pre- and postconditions based on the "design by contract" [3] methodology.

References

[1] Object Management Group. Model Driven Architecture — A Technical Perspective, September 2001. (`http://www.omg.org`).

[2] VIATRA2 Model Transformation Framework. An Eclipse GMT Subproject (`http://www.eclipse.org/gmt/VIATRA2/`).

[3] Bertrand Meyer: Object-oriented software construction (2nd ed.). Prentice-Hall, Inc., 1997.