KandISTI: A Family of Model Checkers for the Analysis of Software Designs

by Maurice ter Beek, Stefania Gnesi and Franco Mazzanti

Driven by a series of European projects, researchers from the Formal Methods and Tools lab of ISTI-CNR have developed a family of model-checking tools for the computer-aided verification of the correctness of software designs. To date, these tools have been applied to a range of case studies in the railway, automotive and telecommunication fields.

When analyzing the correctness of designs in complex software systems during their early stages of development, it is essential to apply formal methods and tools. The broader system is described using a formal specification language and its relative correctness (with respect to relevant behavioural properties) is checked by formally evaluating temporal logic formulas over the underlying computational model. Over the last two decades, we have developed the KandISTI family of model checkers, each one based on a different specification language, but all sharing a common (on-the-fly) temporal logic and verification engine.

The main objective of the KandISTI framework is to provide formal support to the software design process, especially in the early stages of the incremental design phase (i.e., when designs are still likely to be incomplete and likely to contain mistakes). The main features of KandISTI focus on the possibilities of (i) manually exploring the evolution of a system and generating a summary of its behaviours; (ii) investigating abstract system properties using a temporal logic supported by an on-the-fly model checker; and (iii) obtaining a clear explanation of the model-checking results, in terms of possible evolutions of the specific computational model.

The first tool in the family was the FMC model checker which described a system by a hierarchical composition of sequential automata. This tool proved to be a very useful aid when teaching the fundamentals of automated verification techniques in the context of software engineering courses. As an attempt to reduce the gap between theoreticians and software engineers, the original model-checking approach was experimented over a computational model based on UML statecharts. In the context of the FP5 and FP6 EU projects AGILE and SENSORIA, this has led to the development of UMC, in which a system is specified as a set of communicating UML-like state machines.

In cooperation with Telecom Italia, UMC was used to model and verify an asynchronous version of the SOAP communication protocol and model and analyse an automotive scenario provided by an industrial partner of the SENSORIA project. Currently UMC is being used successfully in the experimentation of a model-checking-based design methodology in the context of the regional project TRACE-IT (Train Control Enhancement via Information Technology). This project aims to develop an automatic train supervision system that guarantees a deadlock-free status for train dispatches, even when there are arbitrary delays with respect to the original timetable. The largest model we analysed in this context had a statespace of 35 million states.

Again in the context of SENSORIA, we developed the CMC model checker for the service-oriented process algebra COWS. Service-oriented systems require a logic that expresses the correlation between dynamically generated values appearing inside actions at different times. These values represent the correlation values which allow, e.g., to relate the responses of a service to their specific requests or to handle the concept of a session involving a long sequence of interactions among interacting partners. CMC was used to model and analyse service-oriented scenarios from the automotive and finance domains, as provided by industrial partners in the project.

The most recent member of the KandISTI family is VMC, which was developed specifically for the specification and verification of software product families. VMC performs two kinds of behavioural variability analyses on a given family of products. The first is a logic property expressed in a variability-aware version of a known logic. This can directly be verified against the high-level specification of the product family behaviour, relying on the fact that under certain syntactic conditions the validity of the property over the family model guarantees the validity of the same property for all product models of the family. The second is that the actual set of

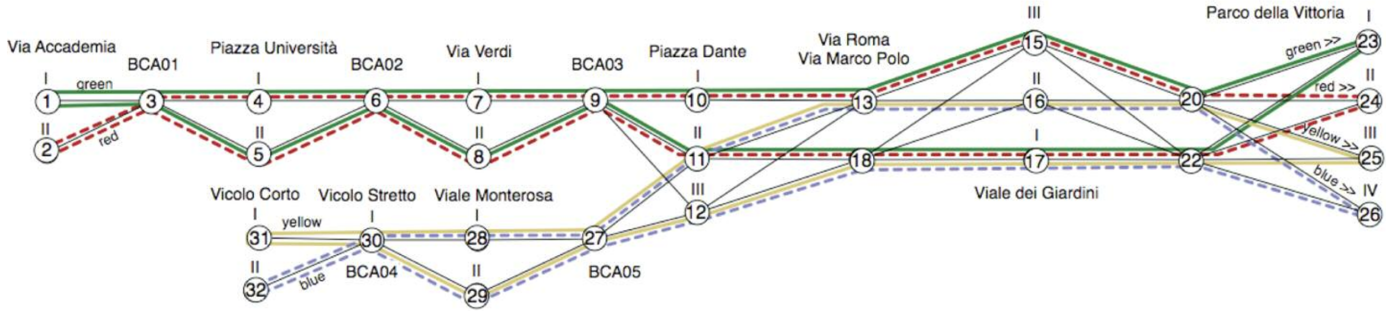

Figure 1: The railway yard layout and missions for trains on the green, red, yellow and blue lines.

valid product behaviours can be generated explicitly and the resulting specifications can be verified against the same logic property (this is surely less efficient than direct verification, but it makes it possible to identify precisely why the original property failed over the whole family). Experimentation with VMC is on-going in the context of the EU FP7 project QUANTICOL. To date, only a small version of the bike-sharing case study from QUANTICOL has been considered but more effort is needed to evaluate VMC on more realistically sized problems.

Link: http://fmt.isti.cnr.it/kandisti

References:
[1] M.H. ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti. "A state/event-based model-checking approach for the analysis of abstract system properties." Science of Computer Programming 76(2):119-135, 2011 http://dx.doi.org/10.1016/j.scico.2010.07.002
[2] A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese, and F. Tiezzi. "A logical verification methodology for service-oriented computing." ACM Transaction on Software Engineering and Methodology 21(3):16, 2012. http://doi.acm.org/10.1145/2211616.2211619
[3] F. Mazzanti, G. O. Spagnolo, and A. Ferrari. "Designing a deadlock-free train scheduler: A model checking approach." NASA Formal Methods, LNCS 8430, Springer, 2014, 264-269. http://dx.doi.org/10.1007/978-3-319-06200-6_22

Please contact:
Franco Mazzanti, ISTI-CNR, Italy
E-mail: franco.mazzanti@isti.cnr.it

QVTo Model Transformations: Assessing and Improving their Quality

by Christine M. Gerpheide, Ramon R.H. Schiffelers and Alexander Serebrenik

Applying a unique bottom-up approach, we (Eindhoven University of Technology (The Netherlands)) worked collaboratively with ASML, the leading provider of complex lithography systems for the semiconductor industry, to assess the quality of QVTo model transformations. This approach combines sound qualitative methods with solid engineering to proactively improve QVTo quality in a practitioner setting.

Model-driven engineering (MDE) can be used to develop highly reliable software which offers a range of benefits from systems analysis and verification to code generation. In MDE, system models are created by domain experts and then transformed into other models or code using model transformations. One of the languages used for writing these model transformations is QVT Operational Mappings (QVTo) which was specified in the 2007 Object Management Group (OMG) standard for model-to-model transformation languages. QVTo is regularly used by both academics and industry practitioners, including ASML, the leading provider of complex lithography systems for the semiconductor industry. Currently ASML has more than 20,000 lines of QVTo code, supporting more than a hundred model transformations.

Despite its widespread use, however, QVTo is a relatively new language. For general-purpose languages, developers have had time to share best practices and establish a number of standard reference points against which the quality of a piece of code can be judged. These are yet to be developed for QVTo. Moreover, QVTo has a large amount of language-specific constructs which are not available in general-purpose languages or even in other model-to-model transformation languages. In fact, QVTo specifications have been described by some as "rather voluminous" and even "fantastically complex". Therefore, it is unclear whether established intuitions about code quality apply to QVTo and a lack of standardized and codified best practices is recognized as a serious challenge in assessing their transformation quality.

In a response to this challenge, ASML and Eindhoven University of Technology have joined in an ongoing collaboration to investigate the quality of QVTo transformations. In addition to assessing the quality of a transformation, this project is also seeking to promote the creation of higher-quality transformations from their inception, improving quality proactively [1].

To achieve this goal, a bottom-up approach was used which combined three qualitative methodologies. To begin, a broad exploratory study which included the analysis of interviews with QVTo experts, a review of the existing literature and other materials, and introspection were completed. Then, a QVTo quality model was developed to formalize QVTo transformation quality: this model consisted of high-level quality goals, quality properties, and evaluation procedures. The quality model was validated using the outcomes from a survey of a broad group of QVTo developers in which they were asked to rate each model property on its importance to QVTo code quality.

Many of the model properties recognized as important for QVTo transformation quality are similar to those in traditional languages (e.g., small function size). However, this analysis also highlighted a number of properties which are specific to QVTo or other model transformation languages. For instance, we found that the following QVTo-specific properties were considered important for quality: the use of only a few black boxes, few queries with side effects, little imperative programming (e.g., for-loops) and small init sections. Deletion using trash-bin patterns was also found to be beneficial for per-