A Framework for Formal Verification of Real-Time Systems

Tamás Tóth

Deptartment of Measurement and Information Systems
Budapest University of Technology and Economics,
Budapest, Hungary
totht@mit.bme.hu

Abstract. Formal methods have an important role in ensuring the correctness of safety critical systems. However, their application in industry is always cumbersome: the lack of experts and the complexity of specification languages prevent the utilization of formal verification techniques. In this work we take a step in the direction of making formal methods applicable by introducing a framework that enables efficient modeling and analysis of real-time systems.

1 Introduction

Nowadays, an ever increasing number of information systems are embedded systems that have a dedicated function in a specific, often safety critical application environment (e.g., components of a railway control system). In case of safety critical systems, failures may endanger human life, or result in serious environmental or material damage, thus ensuring conformance to a correct specification is crucial for their development.

To guarantee that a system operates according to its specification, formal verification techniques can be used. These techniques are based on formal representation of both systems and their properties (requirements), which makes it possible to apply mathematical reasoning to investigate their relationship. Moreover, these methods allow verification of systems in an early phase of the development life cycle.

Since behavior of safety critical systems is often time dependent, the notion of time has to be represented in their models. The most prevalent way to model timed systems is the formalism of timed automata. However, this formalism is only suitable to describe timed behavior with respect to constant values, thus its expressive power is not sufficient to model systems with parametric behavior. Parametric timed automata, an extension of the original formalism, addresses this problem.

In this work we introduce a formal modeling framework for supporting the efficient development of parametric timed automaton based formal models. The modeling language is essentially based on the language of the well-know

Symbolic Analysis Laboratory (SAL) framework[1] with extensions to simplify the work of the modelers. These extensions enable the modeling of time dependent behavior on language level.

2 Related Work

Our work is inspired by the *SAL* model checker [5] and its language (our extensions are introduced in Section 3). The SAL language enables compact modeling of systems in terms of (unlabeled) symbolic transition systems, however it doesn't support explicit modeling of time related behavior. The aim was to preserve compatibility so that the timed models of our extended language can still be transformed to the input of SAL. As another related tool, UPPAAL [1] is a model checker widely used for the verification of timed systems. It has a graphical interface and it provides efficient model checking algorithms to verify timed automata. UPPAAL models can also be transformed to our language with some restrictions: our formalism does not handle complex function declarations. Our approach has different strengths as the underlying *Satisfiability Modulo Theories* (SMT) technologies are efficient for even complex data structures of the modeled systems. In addition, complex synchronization constraints can be compactly expressed in our approach. The industrial case study we use was first introduced in [6], where the SAL model checker was used for the verification. Our work now is based on the lessons learnt from that work.

3 The Specification Language

The core of the specification language is a constraint language that enables declaration of uninterpreted constant symbols of complex data types and constraints over them. Supported data types include boolean, integer, real, enumeration, function, array, tuple and record types, and subtypes that are constrained by some formula. Constraints are sorted first order logic formulae in the combined theory of integer and real arithmetic with uninterpreted constant and function symbols.

The specification language enables the modeling of real time systems by means of symbolic transition systems with clock variables and parameters. The definition of a system consists of variable declarations, invariant and urgency constraints that constraint the traces of the system, and the description of the initialization and transition relations of the system with guarded commands. Systems can be composed synchronously or asynchronously to define more complex behavior. Properties are CTL^* formulae over system variables.

For a more detailed description we refer the reader to [7].

[1] `http://sal.csl.sri.com/`

4 The Verification Workflow

The semantics of the language is provided by a series of simplifying model transformations, and a mapping to an SMT problem. The starting point of the workflow is a model in the above language given either directly, or as a result of a transformation from other timed formalisms, e.g. *UPPAAL* [1].

As a first step, the system is automatically flattened, that is, the result of a synchronous, respectively asynchronous composition is established. This is performed by merging the variables, invariant and urgency constraints, and initialization and transition definitions of the components. During this step, many of the constructed transitions can be eliminated by simply checking the satisfiability of their guards with a call to the underlying SMT solver [4].

In the next step, the model is automatically "untimed" by expressing the semantics of delay transitions explicitly. Here, a combined transition semantics [3] [5] is considered, where a transition merges the effects of a delay transition, followed by a discrete transition. For that purpose, a new input variable d is introduced to represent time delay. Such an untimed system model can easily be mapped to SAL or other intermediate formalisms. At the same time, transition (and initialization) definitions are completed, that is, assignments for unmodified variables (e.g., variables of asynchronous components) are made explicit. As a result, each variable (even those of some complex data type) is assigned a value in at most one assignment of a behavior definition.

The symbolic transition system represented by the model can then be easily expressed in SMT by transforming initialization and transition definitions of the system to predicates $I(\bar{x})$, respectively $T(\bar{x}, \bar{x}')$ as usual [2].

The tooling is implemented in *Eclipse*[2] using *Eclipse Modeling Framework*[3] and relating technologies.

- *Abstract syntax.* The abstract syntax of the language is implemented as a metamodel in *EMF*. It is defined as an extension of the core language suitable for defining complex data types and expressions.
- *Concrete syntax.* The textual concrete syntax is defined by an LL^*-parsable grammar. The textual editor is then generated using the *Xtext*[4] tooling.
- *Semantics.* Model transformations that define the semantics of the language are implemented in *Xtend*[5].
- *Well-formedness rules.* Together with other validation constraints, algorithms for type checking and type inference are implemented for the type system of the language.

[2] `http://eclipse.org/`
[3] `http://eclipse.org/modeling/emf/`
[4] `http://eclipse.org/Xtext/`
[5] `http://eclipse.org/xtend/`

5 Evaluation and Conclusion

This work is one step towards scalable formal modeling. We proposed a modeling language to provide better support for the designers of formal models by focusing on the aspects of data semantics, time dependent behavior, parametrization, and synchronous and asynchronous composition of components. Instead of manually coding flat transition systems, we provided automated model transformations from our extended language to more simple transition systems that can be directly mapped to the input of existing SMT solvers. This way and automated verification workflow is offered.

To evaluate the effectiveness of our language, the formal model of the introduced case study was developed in both our and the SAL languages. Comparing the results, the complexity of the developed models are the following.

The SAL model contains:

- 5 components for modeling the basic behavior consisting of 410 lines of code,
- 2 components for supporting the proper analysis of the temporal logic specification consisting of 105 lines of code,
- 3 components for recognizing the loops in the state space (required for the verification) consisting of 145 lines of code.

The formal model in our extended language contains 4 components and 235 lines of code, which demonstrates that the new language has its advantage. Moreover, it does not require additional components for the analysis.

Compared to UPPAAL timed automata, the main advantage of our language is the greater flexibility in the handling of clock variables and clock constraints. However, as UPPAAL provides a graphical modeling interface, in case of small models it makes the development of formal models more simple. Regarding the efficiency of verification, both approaches have their strengths.

In the future we plan to further improve our language with higher level modeling constructs. We also plan to develop new model checking algorithms based on induction techniques.

References

1. Behrmann, G., David, A., Larsen, K.G., Möller, O., Pettersson, P., Yi, W.: Uppaal - present and future. In: Proc. of 40*th* IEEE Conference on Decision and Control. IEEE Computer Society Press (2001)
2. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design 19(1), 7–34 (2001)
3. Kindermann, R., Junttila, T., Niemel, I.: Smt-based induction methods for timed systems. In: Jurdziski, M., Nikovi, D. (eds.) Formal Modeling and Analysis of Timed Systems, LNCS, vol. 7595, pp. 171–187. Springer Berlin Heidelberg (2012)
4. Moura, L., Björner, N.: Z3: An efficient smt solver. In: Ramakrishnan, C., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol. 4963, pp. 337–340. Springer Berlin Heidelberg (2008)

5. Pike, L.: Real-time system verification by k-induction. Tech. Rep. TM-2005-213751, NASA Langley Research Center (May 2005)
6. Tóth, T., Vörös, A., Majzik, I.: K-induction based verification of real-time safety critical systems. In: Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., Kacprzyk, J. (eds.) New Results in Dependability and Computer Systems, Advances in Intelligent Systems and Computing, vol. 224, pp. 469–478. Springer International Publishing (2013)
7. Tóth, T., Vörös, A., Majzik, I.: Verification of a real-time safety-critical protocol using a modelling language with formal data and behaviour semantics. In: Bondavalli, A., Ceccarelli, A., Ortmeier, F. (eds.) Computer Safety, Reliability, and Security, Lecture Notes in Computer Science, vol. 8696, pp. 207–218. Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-10557-4_24