Formal Verification of Discrete Event Model[*]

Zhihao Lu
Information Engineering College
Capital Normal University
Beijing China
2252029657@qq.com

Rui Wang
Information Engineering College
Capital Normal University
Beijing China
rwang04@cnu.edu.cn

Yong Guan
Information Engineering College
Capital Normal University
Beijing China

ABSTRACT

Ptolemy is a modeling and simulation toolkit widely used in cyber physical systems. Formal verification is a very important method to guarantee the correctness of systems. However, the proposed verification approach in Ptolemy is inconvenient and only supports a few actors. In this paper, we present a model translation based approach to verify the Ptolemy Discrete-Event model and implement a plug-in tool in Ptolemy. A set of mapping rules are designed to translate the Ptolemy Discrete-Event model into a network of timed automata. A template library that translates from actors in Ptolemy Discrete-Event model is also built in advance. A case study of traffic lights control system has been successfully translated and verified. The verification results are reliable and valid. The experimental results confirm our approach is useful to verify the Ptolemy Discrete-Event model.

CCS CONCEPTS

• **Security and privacy → Formal security models; Logic and verification**

KEYWORDS

Formal verification, Ptolemy DE model, Automatic model translation, Timed automata

ACM Reference format:

Zhihao Lu, Rui Wang, and Yong Guan. 2020. Formal Verification of Discrete Event Model. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Testing, Analysis, and Verification of Cyber-Physical Systems and Internet of Things (TAV-CPS/IoT'20). July 19, 2020, Virtual Event, USA. ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/3402842.3407159

[*]This work was supported by NSFC 61877040

1 INTRODUCTION

Ptolemy II[1] is an open-source software package for modeling and simulating concurrent, realtime and embedded systems. It facilitates the design process with a component assembly framework and a graphical user interface[2]. The provided actor library, which is extendible, allows the designers to construct models quickly. Systems in Ptolemy II focus on assembly of concurrent components which are called actors. The key principle of Ptolemy model is the use of well-defined models of computation that govern the interaction between actors. Ptolemy II supports many models of computation and discrete event(DE) is one of them.

However, Ptolemy can't verify model[3]. Now Ptolemy DE model is used widely, especially in aerospace, military, medical and autopilot. More and more practitioners choose Ptolemy DE model of computation. In these highly reliable areas, it will be dangerous if the model is not verified. In order to solve the predicament, we present an automated model translation approach for verifying Ptolemy DE model in this paper. We choose the Uppaal[4][5] tool to verify the translation model because of the advantage of Uppaal in modeling, simulation and verification. It can analyze the timing behavior of a system, and methods for checking both safety and liveness properties of time automata are well developed and intensively studied in Uppaal. The tool contains the editor, simulator and verifier, which can parse the translated model file, simulate the model to check equivalence with source model and verify properties of model. So, Uppaal is well suited to the validation of Ptolemy DE model. Mapping rules are designed to translate the Ptolemy DE model into a network of timed automata in Uppaal. A template library that translates actors in Ptolemy DE is also built in advance. The approach first parses Ptolemy DE model file and generates an XML file that can be parsed in Uppaal. We also develop a plug-in tool based on this approach and integrate it into Ptolemy environment. The main contributions of this paper are as follows:(1) we propose a new model translation based approach to verify Ptolemy DE model. A translation algorithm from Ptolemy DE model to Uppaal timed automata is defined that keeps the architecture consistency of the two models. The concrete workflow consists of loading defined mapping rules and template library, parsing Ptolemy DE model, automatically generating Uppaal model code, verifying properties and experimental implementation. (2) We develop a plug-in and integrate it into the Ptolemy tool to support the verification of DE model. The model can be translated and verified in Ptolemy.

2 MAPPING RULES

An actor consists of some built-in parameters, several input or output ports and special function in Ptolemy DE model. In our approach, each actor is translated into an automaton depending on an automaton template. These templates in template library are essentially timed automata and we design at least a timed automaton template for every actor according to the its function. The concrete contents about ports, parameters of actor are mapping into the template by mapping rules and make templates to be real timed automata. A DE model consists of a lot of actors so that there are many timed automata are generated. These timed automata communicate with each other by synchronization.

2.1 Mapping Rule Formulation

In our approach, the key is the the mapping rules formulation.

2.1.1 Mapping Rule of DiscreteClock Actor: DiscreteClock is a versatile event generator. It has three noticeable parameters: *period, offsets and values.* And its simplest use is to generate a sequence of events that are regularly spaced in time. Its default parameters cause the actor to produce tokens with value 1 (an int) spaced one time unit apart, starting at execution start time (typically time zero). But DiscreteClock can generate much more complex event patterns that cycle through a finite sequence of values and have periodically repeating time offsets from the start of each period.

$$s0 \xrightarrow{getI\_update} s1 \xrightarrow[po\_update]{\substack{guard\\ sync!}} s0$$

$$s1 \xrightarrow[iclock\_update]{!guard} s1$$

The first edge *s0->s1* is used to get values of parameter *period, offsets and values.* We define a data typed int to simulate clock, the second edge *s1->s1* is used to the clock accumulation, the third edge *s1->s0* is used to send a synchronization and update the parameters and output port value if the *guard* is true. The *guard* guarantees that the automaton will send a synchronization at a particular clock.

2.1.2 Mapping Rule of FSMActor: FSMActor is an actor that can create a finite state machine (FSM). The FSM is built using states and relations rather than actors and connections. A relation represents the act of moving from one state to another, it can be triggered by a guard(g), which specifies the conditions under which the transition is taken. It is also possible to specify outputActions (o, actions that produce outputs when the relation is taken) and setActions (s, actions that set parameters when the relation is taken). FSMActor consists of multiple transitions, but the mapping rules of transitions are the same. i.e. $s0 \xrightarrow{g,o,s} s1$,the mapping rule is shown below:

$$s0 \xrightarrow[ip\_update]{synci?} s0p \xrightarrow[po\_update]{guard} s0o \xrightarrow{synco!} s1$$

The first edge *s0->s0p* is used to receive synchronization, and reset the value of *ip_isPresent* is true. The second edge *s0p->s0o* is used to update parameters and the values of output ports if the *guard* is true. The third edge *s0o->s1* is used to send synchronization. If the transition *s0->s1* in FSMActor doesn't need an input event, *s0* goes straight *s0o* with *guard* and *po_update.* And if the transition products more than one output event, an another edge *s0o->s0o* is needed to send an another synchronization.

2.1.3 Mapping Rule of TimedDelay Actor: The actor increases the model time of the input event by a specified amount. The increment is parameter of actor and its amount is displayed in the icon and defaults to 1.0. This actor delays an event in model time.

$$s0 \xrightarrow[update]{synci?} s1 \xrightarrow[update]{guard} s0$$

The first edge *s0->s1* is used to receive synchronization and reset the local clock to 0, the second edge *s1->s0* presents it sends a synchronization update the output port value when the local clock arrivals the delay clock.

2.2 Translation Correctness

We define a DE model and its associated timed automata are equivalent if their observable execution are equivalent, i.e. (1)the order that actor fires is the same as the order that automaton runs, (2)the two model have the same execution path if the input settings are the same, (3)all variable values at each execution step of the two model are equal.

3 CONCLUSION

In this paper, we proposed a model translation based approach to automatically translate and verify the Ptolemy DE model. Since Uppaal tool has a verifier and can effectively express discrete events, we translated a Ptolemy DE model to a network of timed automata in Uppaal and verify them using Uppaal tool. We implemented the approach and developed a plug-in in Ptolemy. In the future, we will extend the verification application area by enriching template library and design a method to automatically detect the consistency of the Ptolemy DE model and the network.

REFERENCES

[1] Lee, Edward A., 2006, Modeling, Simulation, and Design of Concurrent Real-Time Embedded Systems Using Ptolemy.
[2] Ht, L Ig , 2001, Overview Of The Ptolemy Project, Kidney International.
[3] Chihhong, 2008, Applied verification: The ptolemy approach.
[4] Bengtsson, 2006, Timed Automata: Semantics, Algorithms and Tools,
[5] Kim G, 2005, Tool Environment for Validation and Verification of Real-Time System.