Modeling Gear Control System using UPPAAL*

16340154 – Nino Lau – Group 14

School of Data and Computer Science, Sun Yat-sen University

Dec. 22nd , 2018

In the control system of modern cars, the ubiquious nature of gear-shifting operation has driven us to explore an effective method to check the correctness of the model. Inspired by Magnus Lindahl *et al.*, we consider using **UPPAAL***, a powerful tool kit providing reachability analysis, to describe our gear control system. Our verification results prove the correctness of the full design, without any additional effort to implement the existing model-checker.

I. Introduction

Aim to design and analyze the industrial gear control system, we conducted our experiment using **UPPAAL*** reachablity analysis in combination with a certain syntactical manipulation. This section will show some key information about our system design.

System Design

The gear control Interface provides shared boundary for information exchange between gear control system and its users, receiving service requests and keeping information about the current status.

The gear-requests from the users are delivered over a communication network to GearControl. The GearControl accomplishes a zero torque transmission, preparing to release the currently set gear. Then it achieves synchronous speed over the transmission and sets the new gear. Once the gear is set, the engine torque is increased back to previous level. In this way, the gear change is achieved.

Gears are changed by requesting services provided by the components in their environment. Specifically, the GearBox is responsible for setting or releasing a gear in the given time limit; the Clutch can open or close, used when the zero torque or synchronous speed not possible; the Engine offers three modes of torque control, which have different angular accelerations to deal with various conditions.

Lock Free

A dead-lock is a state in which each member of a group of actions, is waiting for some other member to release a lock. Generally, dead-lock is generated when such specific conditions *i.e.* mutual exclusion, hold and wait, no preemption, circular wait are met.

Component with a unrecoverable error cannot recover its error by itself. In this system, unrecoverable error happens in `COpenError`, `CCloseError`, `GNeuError` and `GSetError` location, because states have no where to go in these locations. While recoverable error is detected in the location `CheckTorque` and `CheckSynSpeed`, meaning that we do have possibility to recover the errors in these locations. In our implementation, integer variables `ErrStat` and `UseCase` are used to represent unrecoverable and recoverable error types. Our design ensures that dead-lock and live-lock are free in our design, and the functionality and predictability requirements are both successfully satisfied.

Real-time Transition

The bounded response time property puts forward a requirement for the system to respond in a specific time to ensure the safe operation of the system. In the bounded response time properties formula $f_1 \leadsto_{\leq T} f_2$, f_1, f_2 are boolean combinations of atomic propositions representing request and response respectively. f is an invariant property used to express safety properties, a system satisfy $Inv(f)$ if all its reachable states satisfy f. Interestingly, $f_1 \leadsto_{\leq T} f_2$ is tentatively closed to the strong until operator in LTL with an additional time bound (but f_1 does not need to be continuously true here), meaning that f_2 must be true within T when f_1 is true. Shown in **Experiment**, with the addition of boolean variables v_n and their corresponding clocks c_n, all of the bounded response time properties are fully satisfied by our design.

II. Method

To verify bounded response time properties, we introduce some boolean variables and corresponding timers to provide reset options. After verification, we found that our system fully satisfies bounded response time properties, that is, state transitions in our system are real-time (with little time delay).

Project Experiment

A detailed illustration about bounded response time properties implementation is given using the example in original paper **Table. 2** (6).

$$\mathsf{GearBox} \ @\ \texttt{ErrorNeu} \leadsto_{\leq 200} \mathsf{GearControl} \ @\ \texttt{GNeuError}$$

A state transition in GearBox from `Opening` to `ErrorNeu` is reflected in upper layer by the state transition towards `GNeuError` in GearControl. In order to verified such property, our extension work is to make sure that once the GearBox state transition is complete, the GearControl state transition is going to be finished within 200 milliseconds.

Using invariant condition method to verify, we introduce a boolean variable v_6 and a corresponding timer c_6 into our design. Here, v_6 and $\neg v_6$ serve as guards of `ErrorNeu`. We copy the existing edge and assign operation $v_6 := true, c_6 = 0$ to the reset edge. After that, $v_6 := false$ is set for all edges towards `GNeuError` in GearControl. When the system works, this design will ensure time-limited state transitions. In the example above, our goal is to verify `Inv`$(v_6 \Rightarrow c_6 \leq 200)$. when the model goes into `ErrorNeu` location, $v_6 = 0, c_6 = 0$. Then, $v_6 = true$ until the model goes into `GNeuError` location. Once $v_6 = true$, we can infer that $c_6 \leq 200$ during this period.

It is worth noting that invariant condition is not the only way to verify bounded response time properties. We found that we can also verify bounded response time properties in suprema formula formatted like `sup(condition):expression`. In other words, it is equivalent to use `sup(`v_6`):`c_6 in this instance, which means that $v_6 = true$ implies $c_6 \leq$ UP . We add an edge from `GNeuError` to itself and add a timer c_6 on this self-edge. Verification result gives an upper bound (200 msec) for timer c_6 as expected. Considering that there are only upper bounds for transition delay (no lower bounds, *i.e.* LOW), no verification form like `inf(condition):expression` are involved in our design.

Similarly, we use these methods to ensure the other transitions are real-time in our system.

Experience and Lesson

This experiment brings us many ideas and inspirations. **Definition 4.** symbolizes the definition of states and their transitions of a real-time systems, providing us new ideas to build automaton from atomic propositions and data variables to trace formula. **Section 6.2.** uses bounded response time properties to describe requirements in the whole system, leading us to think about how to solve problems with real-time software thinking. **Fig. 3.** creates a decorated version of automaton, opening up new roads for us to consider how to verify the synchronization of various components in different layer.

Apart from the knowledge level, this practical training also bring me a new understanding of teamwork: use my own actions to inspire instead of complaining; accept other works of very different styles; establish common goals although everyone has different focus...

To sum up, this practical training on implementing software system not only enables us to have a deeper understanding of the knowledge we have learned in our course, but also enables us to get to know how each part works, how to formulate practical problems, how to design verification and so on.

III. Conclusion

We have implemented an industrial case of gear control system design. The reachability analysis function of **Uppaal*** tool kit is utilized to ensure that our system meets the performance, predictability, functionality and error detection requirements. Through this experiment, we put the knowledge into practice, and our cognition of real-time software system is strengthened.

IV. Acknowledgement

- First and foremost, I would like to show my deepest gratitude to my teacher, professor *Kangeu*, who has provided us with valuable theoretical guidance of real-time system. Without her enlightening instruction, impressive kindness and patience, we could never have completed our projects. Her keen and vigorous academic observation enlightens me not only in this thesis but also in my future study.
- My sincere appreciation also goes to my *teammates* in our group. We got along in harmony, accomplished tasks together, and each brought out his own strengths. All of these make this teamwork-experience valuable.

V. Resources

View the full implementation on my GitHub: `https://github.com/LovelyBuggies/Uppaal_GearControlSystem`.