Illustration and Evaluation of the Given Gear Controller

Johnson LiShuailin

Abstract. In this paper, we illustrate an example of gear controller, along with evaluation on this given example. This control system is composed of 5 parts, namely interface, gear controller, gear box, clutch and engine. The interface interacts with the user, while gear controller manipulates the rest three components. Besides these, we also report on the problem of deadlock——the two circumstances of deadlock and the method of checking bounded time response of properties which is supremum and infimum. The report will also contain the process of exploring the project, the progress made by me and my group mates.

1 Introduction

This project, in fact, is a joint project between industry and academia. It includes mainly three parts: the implementation of the template, the verification of the properties and the bounded time response check. The project contains five templates as said, sixteen properties to verify and a method to check bounded time response.
Our group firstly implement the templates and reorganize them into one complete project. During this process, we've encountered some problems like syntactical issue and deadlock problem. We refer to deadlock free, which is the liveness, as A[ ] not deadlock. However, this is associated with the two definitions of deadlock. Because in refer to the given project, we have some locations in the gear controller template named COpenError or GNeuError which have two entries but no exit. So when an automata enters these locations, it has no other choice but terminate the trace, which according to a tutorial, another definition of deadlock. Here is the other definition: A state is a deadlock state if there are no outgoing action transitions neither from the state itself or any of its delay successors.

2 Experience

In this section, I mainly talk about the experience of our exploring process.

2.1 Verify Properties by using Supremum & Infimum

After solving the syntactical issue and deadlock problem, we move on to verify the properties, with the method of supremum and infimum.
We first add an upper bound as well as a lower bound. We set a property like: E<> ((ErrStat==0 and UseCase==0 and SysTimer<=899 and SysTimer > 150) and not (GearControl.GearChanged)) to check whether it's possible that we don't get a gear change with SysTimer between 150 and 899, which verify the property sixteen shown on the paper.
Besides, we also use supremum and infimum like this: sup { expression } : list. The expression in the list are evaluated only on the states that satisfy the expression (a state predicate) that acts like an observation and the inf formula are similar to sup but for infima. A state predicate should be used when a clock infimum is asked otherwise the trivial result is >=0.

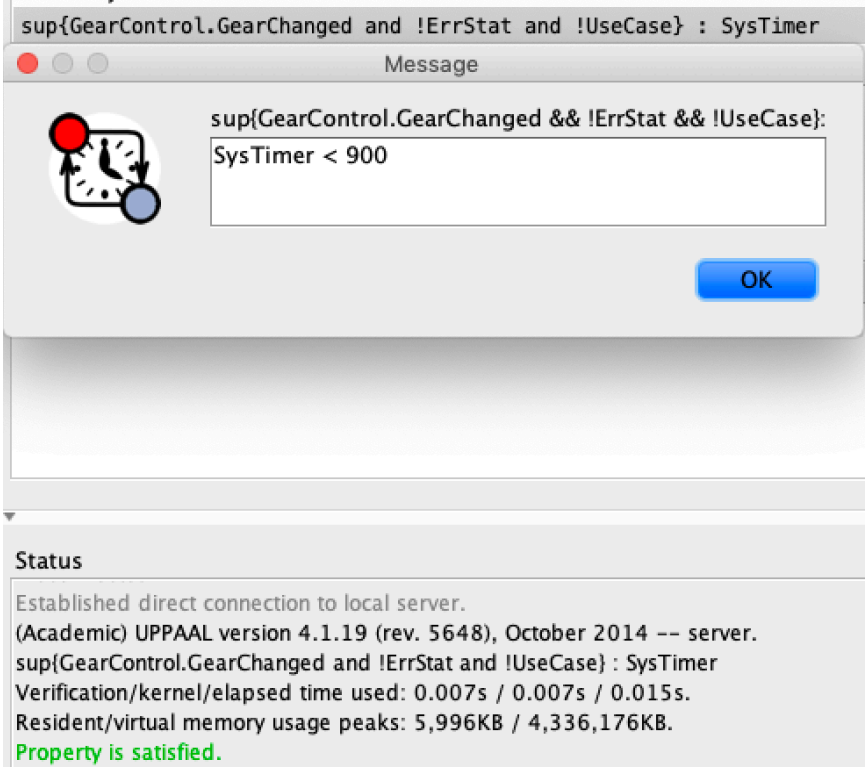

Fig.1 Illustration of the sup & inf use

2.2 Reachability Analysis

During the process of verifying the properties, we also encountered the reachability analysis problem, that is, a solution to the reachability problem in the particular context of distributed systems. It is used to determine which global states can be reached by a distributed system which consists of a certain number of local entities that communicated by the exchange of messages. The following pictures are the required formula and checked properties.

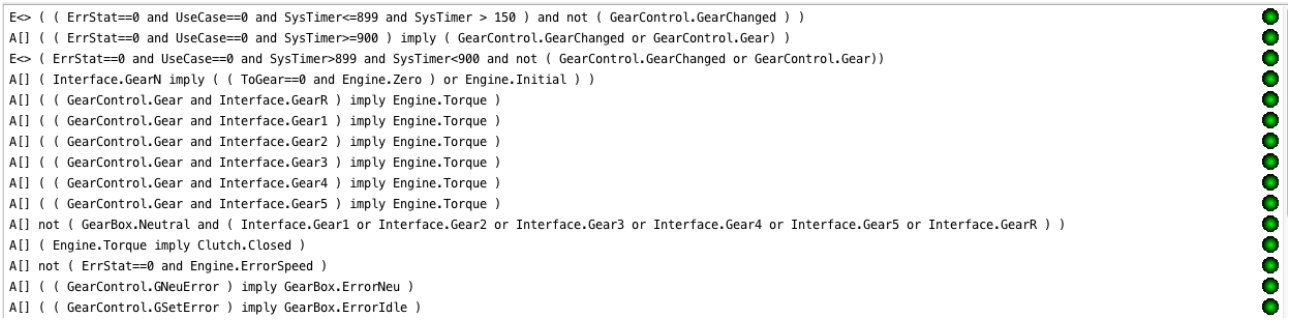

Fig.2 Display of some checked properties

From the picture shown above, we can see how the formula be transmitted to Uppaal understood properties.
In fact, reachability properties are the simplest form of properties. They ask whether a given state formula ϕ, possibly can be satisfied by any reachable state, that is if there exist a path starting at the initial state, such that ϕ is eventually satisfied along the path.

2.3 Explanation of the Given Example

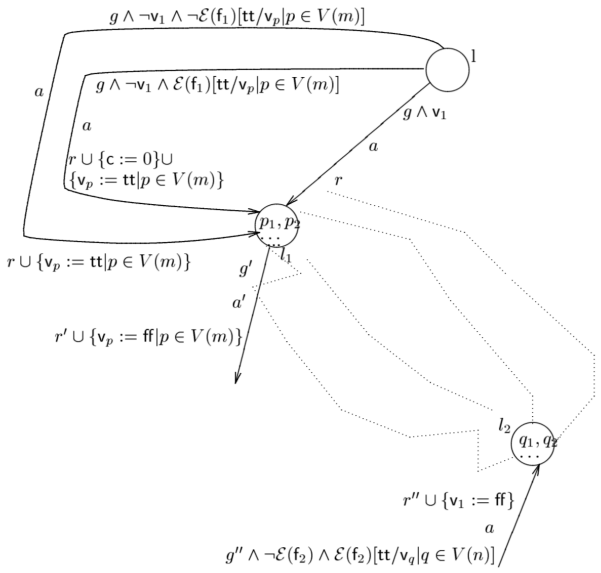

Fig.3 Illustration of the decorated version M(Ai) of Ai

In the given paper, we verify bounded response time properties by reachability analysis. The picture shown above is the decorated version, after the original one and modified one. In comparison with the first two version, this decorated version, we duplicate more edges from location L entering L1. Besides, we also add some boolean variables to denote the truth of some formulas. Also, we add v1 as a guard to the auxiliary copies of the edge entering location L1 as well as v1:=ff as a reset-operation.
In a word, we take methods to make sure that the property v1=>c<=3 is an invariant and there is only one situation that violates the invariant: v1 and c>3.

3 Conclusion

In conclusion, we used the model checker Uppaal, to build up a gear control model and verify the required properties as well as use the method of sup & inf to check the time bounded response properties.
We've also learned some more, i.e. the CCS-like parallel composition operator to denote the network of automata and multiple definitions of deadlock under different circumstances.
And we also have some doubts and questions about the Uppaal using like in the verifier we may pass the property if we check them altogether but when check one by one, some of them may fail. I also noticed that the property which was not satisfied when individually, during the process of checking together, spent a very short time, displayed as 0, but as 0.007s when checked individually.

```
E<> GearBox.ErrorNeu
Verification/kernel/elapsed time used: 0.007s / 0.005s / 0.013s.
Resident/virtual memory usage peaks: 6,288KB / 4,335,908KB.
Property is not satisfied.
E<> GearBox.ErrorNeu
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 6,288KB / 4,335,908KB.
Property is satisfied.
```

Fig.4 Doubt about the Uppaal