

Using Model Checking to Verify the Logic Module of Flight Control Software

Shunkun Yang and Guoqi Li

School of reliability and system engineering, Beihang University,

100191 Beijing, China

ysk@buaa.edu.cn, gqli@buaa.edu.cn

Keywords: model checking, logic module, flight control

Abstract. Model checking is an important method to verify state machine based system. In this paper, we using PAT, a novel and powerful model checking tool, to verify the logic module of flight control software, which is public available. Conclusions are drawn from the verification and these are valuable for similar researches.

Introduction

In computer science, model checking [1] refers to the following problem: given a model of a system, exhaustively and automatically check whether this model meets a given specification. Typically, one has hardware or software systems in mind, whereas the specification contains safety requirements such as the absence of deadlocks and similar critical states that can cause the system to crash. Traditionally, model checking is used to detect safety violations in requirements specifications [2]. Recently, hot topic is using model checking to embedded software [3] and Statecharts, which is one of diagrams of UML [4].

In this paper, we using PAT (Process Analysis Toolkit), which is an enhanced simulator, model checker and refinement checker for concurrent and real-time systems [5], to verify the logic module of flight control software, which is based on Stateflow of Matlab and is public available.

Conclusions are drawn from the verification and these are valuable for similar researches.

Verification

Systems Being Verified

For generalization, the case is selected from a public source, demonstration of Matlab.

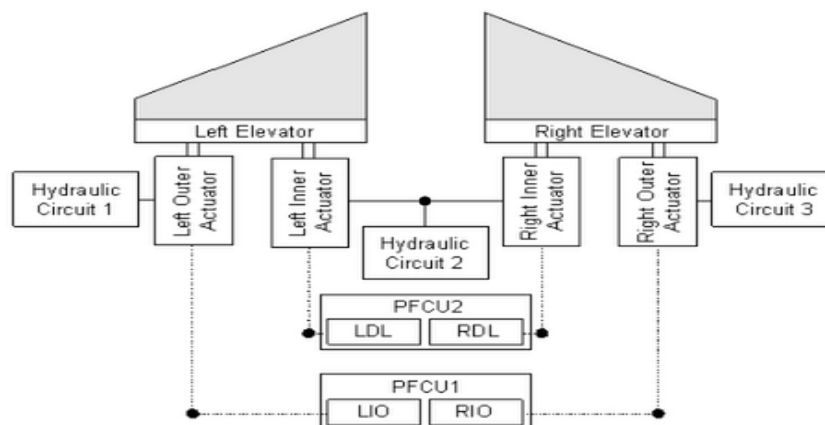


Fig 1. Schematic showing how the components of the elevator system are connected to one another

Aircraft elevator may be the appropriate system for our study. A typical aircraft has two elevators attached on the horizontal tails. And they are distributed on both side of the fuselage named left elevator and right elevator. There are number of redundant parts in the system to enhance safety of the aircraft. As the figure 1 shows the schematic of the components of elevator system are connected to another.

There are two independent hydraulic actuators per elevator, and three separate hydraulic circuits to drive the actuators. PFCU1 and PFCU2 are the two primary flight control units. PFCU1 is connected with the left outer actuator and right outer actuators. PFCU2 is connected with the left inner actuator and right inner actuators. Two control modules per actuator are used to regulate the full range control law and limited/reduced range control law.

For other detailed information, please refer to demonstration of Matlab.

The Statelflow diagram of the control logic module is show in the figure 2.

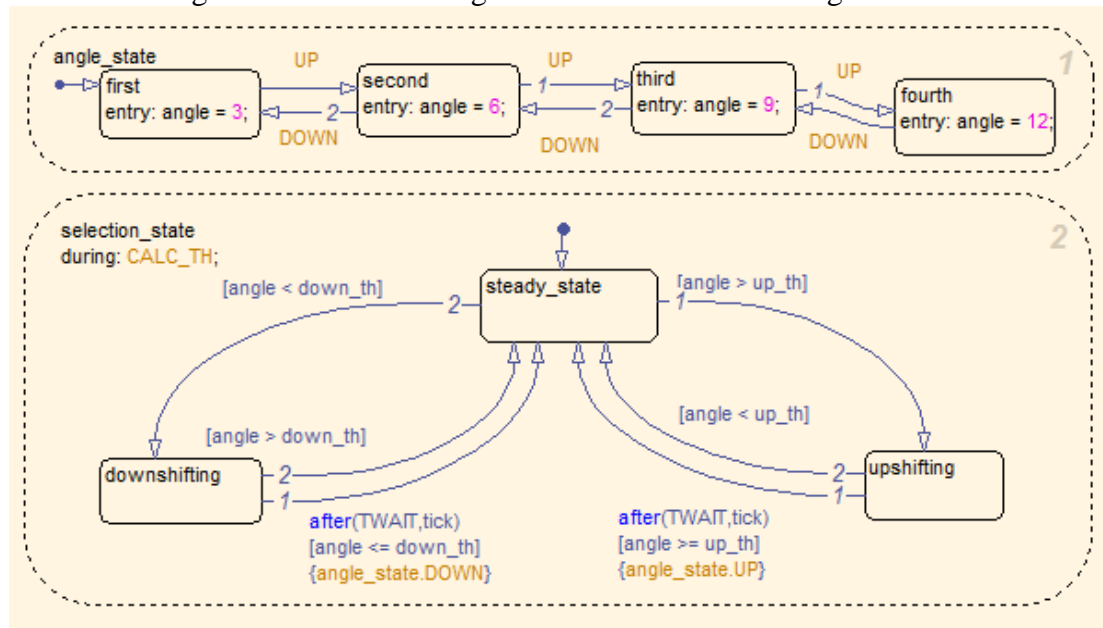


Fig.2. The Statelflow diagram of the control logic module.

In the language of PAT, the first thing to consider, when modeling a system is to point out and describe appropriately all kinds of variables that function in the software system. According to the stateflow, we can clearly figure out the variables and then listing them below using PAT language:

```
var selection_state_downshifting_Status:{0..1} = inactive;
var selection_state_upshifting_Status:{0..1} = inactive;
var angle_state_Status:{0..1} = inactive;
var angle_state_fourth_Status:{0..1} = inactive;
```

The next thing to do is to define the *EnterBu* variable. With the help of the Stateflow chart, we can easily describe those in PAT language:

```
var angle_state_EnterBU = false;
var selection_state_EnterBU = false;
```

There are still other sorts of variables that need to be defined in PAT. Detail is ignored.

After the definition of all kinds of variables in the systems, the next task to complete is describing the transition of the status that take place in the software systems.

Take the downshifting part as an example. To define this part of the software in PAT, the task need to be done is as follow:

1. Describe the event that happens when a status is entered
2. Describe the event that happens during a status
3. Describe the event that happens when existing a status.

The following PAT code is describing the downshifting status:

```
selection_state_downshifting()=
```

```

if (selection_state_downshifting_Status == inactive) {
  selection_state_downshifting_EnAct()
} else {
  if (((angle <= down_th) && (selection_state_downshifting_OUTGOING == 0)) &&
(selection_state_downshifting_Timer >= TWAIT)) {
    {DOWN = occurred;}->angle_state();
  }
}

```

After complete modeling the actuator using PAT, the next thing to do is run the PAT simulation and verification. PAT can verify and check the model with different aspects, such as logical deadlock, and the trace of status. For example, if trace of the variable *angle_state_DurAct* is shown as below:



Fig.3. Creating the trace of variable.

PAT can also check the property of model. We can assert a property using PAT language, run the verification, and PAT will check if the model satisfies the property. If not, the PAT will illustrate with a counterexample.

One of the examples is shown below.

```

#define goal (angle_state_fourth_Status==1 && angle_state_third_Status==1 &&
angle_state_second_Status==1 && angle_state_first_Status==1);
#assert Stateflow()= [] goal;

```

The figure below is the counterexample of this assertion:

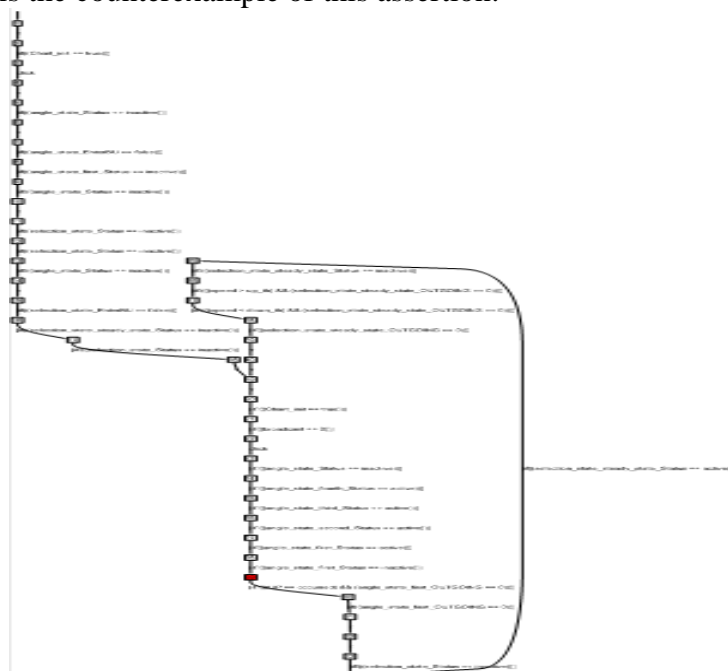


Fig.4. Creating the counterexample using PAT

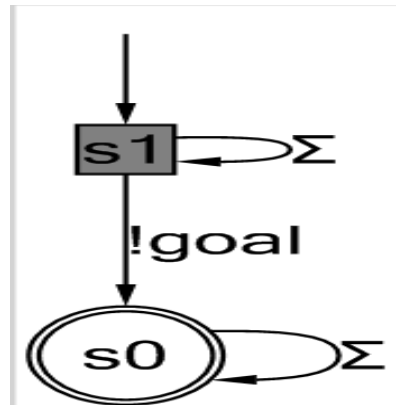


Fig.5. The theory of counterexample

Summary

Model checking is a powerful method to verify state machine based system with low cost and high effective. PAT is also a powerful tool. The workflow and method used in this paper can also be used in similar verification, such as verifying logic module of engine control systems and etc..

There are still some tasks to be done. The first thing is to improve the correctness of the PAT model of the software. For now, we have to translate the mdl to PAT file manually, which is error-prone. The next step of the work includes developing a method to translate the mdl file to PAT automatically. We think the problem we met is general for similar works.

References

- [1] P. Doron, P. Patrizio and S. Paola. Model Checking, Wiley Encyclopedia of Computer Science and Engineering, 2009.
- [2] H. Constance, K.J. James, L. Bruce, A. Myla and B.Ramesh, Using abstraction and model checking to detect safety violations in requirements specifications. J. IEEE transaction on software engineering. 24(1998)927-948.
- [3] C.Lucas, F.Bernd and M.Joao, SMT-based bounded model checking for embedded ANSI-C software. J. IEEE transaction on software engineering. 38(2012)957-974.
- [4] C.William, A.J.Richard, B.Paul, J.H.David, N.David and W.E.William, Optimizing symbolic model checking for statecharts. J. IEEE transaction on software engineering. 27(2001)170-190.
- [5] PAT: Process Analysis Toolkit An Enhanced Simulator, Model Checker and Refinement Checker for Concurrent and Real-time Systems, available at <http://www.comp.nus.edu.sg/~pat/>