**Introduction**

In the context of hardware and software systems, **formal verification** is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.[1] Formal verification is helpful in proving the correctness of many complex systems such as: cryptographic protocols, combinational circuits, digital circuits with internal memory, and software expressed as source code. For the *“elevator system”,* the target is to write model specification of the system and verify that each specification satisfies some desired properties. To learn proper application of the formal techniques and tools to specify and validate given software system is the goal of the project. The report contains the system design, tutorial of the tools used and a discussion of the experience learned.

**Design**

The design mostly contains the states and variables related to the system. The elevator has 3 states, their level: first, second and third. The direction of the elevator also can have three states: down, steady and up. Figure 1 shows the buttons available in the elevator system.

Figure 1: High level design of elevator system

The complete list of button variable is:

* 1. openDoorButton
  2. closeDoorButton
  3. firstButton
  4. secondButton
  5. thirdButton
  6. firstUpButton
  7. secondUpButton
  8. secondDownButton
  9. thirdDownButton

A variable to store the status of the button light for each button was created too. The list goes as follows:

* 1. openDoorButtonLight
  2. closeDoorButtonLight
  3. firstButtonLight
  4. secondButtonLight
  5. thirdButtonLight
  6. firstUpButtonLight
  7. secondUpButtonLight
  8. secondDownButtonLight
  9. thirdDownButtonLight

There are some other variables to contain door related information such doorOpen, doorIsOpen, doorCanClose, remainedAsOpen and doorOpenTimer.

**Safety and liveness property**

For the safety property, “the elevator never moves with the door open” was checked in all the tools and implementation. To check for liveness property, various LTL specification were written and verified; such as if a call is made from a floor, the request is served eventually.

List of some properties in NuSMV:

1.1# ELEVATOR REACHES EVERY LEVELS CORRECTLY UPON CALLING

LTLSPEC G ((system.firstButton = TRUE | system.firstUpButton = TRUE) -> F (system.elevatorLevel = first & system.doorOpen = TRUE))

LTLSPEC G ((system.secondButton = TRUE | system.secondUpButton = TRUE | system.secondDownButton = TRUE) -> F (system.elevatorLevel = second & system.doorOpen = TRUE))

LTLSPEC G ((system.thirdButton = TRUE | system.thirdDownButton = TRUE) -> F (system.elevatorLevel = third & system.doorOpen = TRUE))

--2# SAFETY PROPERTY

LTLSPEC G ( !(system.doorOpen = TRUE & (system.direction = up | system.direction = down) ))

**Tutorial of SPIN**

**Spin** is a popular open-source software verification tool, used for the formal verification of multi-threaded software applications. This open source tool was developed at Bell Labs in the UNIX group of the Computing Sciences Research Center. All the resources related to SPIN are available at <http://spinroot.com/spin/>. [2]

To compile .pml file, SPIN and gcc is needed on the system. For our purpose, we used gcc compiler from cygwin for Windows. The setup is pretty straight forward and can be done by following the website <http://spinroot.com/spin/Man/README.html>. Don’t forget to add the environmental variable, as we are going to use command prompt. Once the system is we can use these three commands to compile and run SPIN.

$ spin -a filename.pml

$ cc -o pan pan.c

$ ./pan

Now, the SPIN uses multiple threads to build the model. A sample code can be:

bool doorOpen = false ;

byte direction = 2;

proctype something(){

do

::

if :: (doorOpen== true) -> direnction = 1

:: else -> skip;

fi

od

}

init

{

run something()

}

The init runs the something() function which continuously checks for the conditions and updated the variable. More such threads can be added to update other values and build the system.

After the system is build, LTL specification can be used to verify it.

// reuqest to level 2 is eventually served

ltl p3 { [] ( (secondButton == true || secondUpButton == true) -> <> (elevatorLevel==2 && doorOpen == true) )}

**Tutorial of Alloy**

**Verification results**

Number of properties was successfully verified by the system developed.

The properties verified for NuSMV:

--1.1# ELEVATOR REACHES EVERY LEVELS CORRECTLY UPON CALLING

LTLSPEC G ((system.firstButton = TRUE | system.firstUpButton = TRUE) -> F (system.elevatorLevel = first & system.doorOpen = TRUE))

LTLSPEC G ((system.secondButton = TRUE | system.secondUpButton = TRUE | system.secondDownButton = TRUE) -> F (system.elevatorLevel = second & system.doorOpen = TRUE))

LTLSPEC G ((system.thirdButton = TRUE | system.thirdDownButton = TRUE) -> F (system.elevatorLevel = third & system.doorOpen = TRUE))

--1.2# DOOR OPENS AND CLOSES AT ALL LEVELS

LTLSPEC G ((system.elevatorLevel = first & system.doorOpen = FALSE & system.openDoorButton = TRUE) -> F (system.elevatorLevel = first & system.doorOpen = TRUE))

LTLSPEC G ((system.elevatorLevel = second & system.doorOpen = FALSE & system.openDoorButton = TRUE) -> F (system.elevatorLevel = second & system.doorOpen = TRUE))

LTLSPEC G ((system.elevatorLevel = third & system.doorOpen = FALSE & system.openDoorButton = TRUE) -> F (system.elevatorLevel = third & system.doorOpen = TRUE))

LTLSPEC G ((system.elevatorLevel = first & system.doorOpen = TRUE & system.closeDoorButton = TRUE) -> F (system.elevatorLevel = first & system.doorOpen = FALSE))

LTLSPEC G ((system.elevatorLevel = second & system.doorOpen = TRUE & system.closeDoorButton = TRUE) -> F (system.elevatorLevel = second & system.doorOpen = FALSE))

LTLSPEC G ((system.elevatorLevel = third & system.doorOpen = TRUE & system.closeDoorButton = TRUE) -> F (system.elevatorLevel = third & system.doorOpen = FALSE))

--1.3# DOOR REMAINS OPEN FOR 5 TIME UNITS

LTLSPEC G ((system.remainedAsOpen = 0) -> (system.doorOpen = FALSE))

LTLSPEC G ((system.remainedAsOpen = 1) -> (system.doorOpen = TRUE))

LTLSPEC G ((system.remainedAsOpen = 5) -> (system.doorOpen = TRUE))

LTLSPEC G ((system.remainedAsOpen = 6) -> (system.doorOpen = FALSE))

LTLSPEC G ((system.elevatorLevel = first & system.doorIsOpen = TRUE & system.closeDoorButton = TRUE) -> F (system.elevatorLevel = first & system.doorOpen = TRUE))

LTLSPEC G ((system.elevatorLevel = second & system.doorIsOpen = TRUE & system.closeDoorButton = TRUE) -> F (system.elevatorLevel = second & system.doorOpen = TRUE))

LTLSPEC G ((system.elevatorLevel = third & system.doorIsOpen = TRUE & system.closeDoorButton = TRUE) -> F (system.elevatorLevel = third & system.doorOpen = TRUE))

LTLSPEC G ((system.elevatorLevel = first & system.doorCanClose = TRUE & system.closeDoorButton = TRUE) -> F (system.elevatorLevel = first & system.doorOpen = FALSE))

LTLSPEC G ((system.elevatorLevel = second & system.doorCanClose = TRUE & system.closeDoorButton = TRUE) -> F (system.elevatorLevel = second & system.doorOpen = FALSE))

LTLSPEC G ((system.elevatorLevel = third & system.doorCanClose = TRUE & system.closeDoorButton = TRUE) -> F (system.elevatorLevel = third & system.doorOpen = FALSE))

--1.4# ALL BUTTON\_LIGHTS WORK PROPERLY

LTLSPEC G ((system.openDoorButton = TRUE) -> F (system.openDoorButtonLight = TRUE))

LTLSPEC G ((system.closeDoorButton = TRUE) -> F (system.closeDoorButtonLight = TRUE))

LTLSPEC G ((system.firstButton = TRUE) -> F (system.firstButtonLight = TRUE))

LTLSPEC G ((system.secondButton = TRUE) -> F (system.secondButtonLight = TRUE))

LTLSPEC G ((system.thirdButton = TRUE) -> F (system.thirdButtonLight = TRUE))

LTLSPEC G ((system.firstUpButton = TRUE) -> F (system.firstUpButtonLight = TRUE))

LTLSPEC G ((system.secondUpButton = TRUE) -> F (system.secondUpButtonLight = TRUE))

LTLSPEC G ((system.secondDownButton = TRUE) -> F (system.secondDownButtonLight = TRUE))

LTLSPEC G ((system.thirdDownButton = TRUE) -> F (system.thirdDownButtonLight = TRUE))

--2# SAFETY PROPERTY

LTLSPEC G ( !(system.doorOpen = TRUE & (system.direction = up | system.direction = down) ))

List of properties verified for SPIN:

//door always close when moving

ltl p1 { [] !((direction == 1 || direction == 3) && doorOpen)};

// reuqest to level 1 is eventually served

ltl p2 { [] ( (firstButton == true || firstUpButton == true) -> <> (elevatorLevel==1 && doorOpen == true) )}

// reuqest to level 2 is eventually served

ltl p3 { [] ( (secondButton == true || secondUpButton == true) -> <> (elevatorLevel==2 && doorOpen == true) )}

// reuqest to level 3 is eventually served

ltl p4 { [] ( (thirdButton == true || thirdDownButton == true) -> <> (elevatorLevel==3 && doorOpen == true) )}

// door eventually closes

ltl p5 { []((doorOpen== true) -> <> (doorOpen==false))}

**Experience Learned**

The most important thing learned was how to design a system so that it can be verifiable. Once the design is done, the learning is mainly tool specific. Team work was something very important for this project.

In SPIN, the main challenge was to understand the syntax. It works in a different way than NuSMV. The LTL specifications were also different.

**Conclusion**

System models are approximate imitations of real-world systems and they never exactly imitate the real-world system. Yet, modeling and verification is increasingly being used to solve problems and to aid in decision-making. Both the developers and users of these modeling tools are benefited by the decision making and information obtained from the results of such models. Building the elevator system model and verifying the specification was a really great way to learn about the tools and the methods behind them.

**Reference**

1. <http://en.wikipedia.org/wiki/Formal_verification>
2. <http://spinroot.com/spin/whatispin.html>