Model Based Design and Verification
KF6009
17036098
Joe Pigford

1. Discussion of the specification.

As one pump cannot run when the other is running the specification contains a statement to ensure that does not happen "A[] (p1.On imply not(p2.On))". This checks that in all possible states in pump one is on then pump two must not be active. This also functions as a check for the reverse, pump two on implying pump one is not, as if pump two is on at any point then pump must be off for this to verify. The requirement is not on over off as there are two possible off states "Off" and "Offline" for each pump. Both satisfying this requirement.

The next requirement "A[] (ctrl.EmergencyShutdown imply (not(p1.On) and not(p2.On)))" states that in all possible "EmergencyShutdown" cases both pumps are turned off. This requirement is included as during an emergency shut down the entire system will be shut down to prevent damage to the system. The pumps being part of the system will need to be shut down to satisfy this requirement. The A[] operator was chosen as the pumps will need to be shut down every time the system enters the "EmergencyShutdown" state. The not "(not(p1.On) and not(p2.On))" is this way in order to ensure both pumps are turned off when the system goes into emergency shut down.

The amount of times a sensor has fail to respond to polls is measured by the variable "failCount". At no point during the systems operation can this go over five. The requirement "A[] (ctrl.failCount <= 5)" checks this is satisfied in all possible states. The A[] operator was chosen as this requirement must be satisfied in all possible states the system can enter. It must be less than five as that is the length of the the sensor is given to respond be being polled again. Another requirement is that if "failCount" reaches five the system must go into emergency shut down as the sensor is assumed to have failed. The statement "A<> (ctrl.failCount == 5 imply ctrl.EmergencyShutdown)" is used to check for this. It states that if failCount reaches five the system will in all cases eventually reach "EmergencyShutdown".

When polling a sensor the system must wait up to 0.5 seconds, represented in this model by the clock "sensorTimer", before taking action. Either re-polling or receiving data from the sensor. The system must not wait more than 5 ticks before re-polling the sensor as this is stated in the requirements. "A[] (ctrl.SteamWait imply ctrl.sensorTimer <= 5)" and "A[] (ctrl.WaterWait imply ctrl.sensorTimer <= 5)" state that for all possible states "sensorTimer" cannot go higher than five meaning that it cannot tick for longer than 0.5 seconds before receiving a reply or re-polling. This is here to ensure that in all possible situations the system does not wait more than five ticks before taking action about either sensor.

There are several requirements to ensure pumps are in the correct states before action is taken based upon sensor feedback. When the water level in the boiler is too low the system tries to correct this by turning a pump on. This is not always doable as pumps may be broken or there may be one already running. In the case of the former the system is locked preventing pumps from coming back online before shutting down. "A[] (ctrl.PumpsOffline imply (p1.Offline and p2.Offline))" is used to verify this stating that for every possible state of "PumpsOffline" both pumps are offline. In the event the water is too low and a pump is running the system is shutdown as it is assumed to be faulty. To turn off the pumps and shut the system down it requires that one of the pumps must be running. This statement , "A[] (ctrl.PumpRunning imply (p1.On or p2.On))" , verifies this. It says that in all cases where the system is in the “PumpRunning” state one of the pumps is turned on. This form has been used as if there are no pumps running when this state is entered the system will deadlock as the edges leaving the state have guards checking pumps are on.

2. Discussion of the design of your model.

The system model uses five templates: the controller, steam sensor, water sensor and the two pumps. The two pumps are identical in function differing only in name in order to make controlling them simpler. An earlier version of the pumps had one template duplicated with a mutex controlling whether they could turn on or not. Turning on acquired the mutex making it so the one could not turn on while the other was running. Due to only allowing one synchronisation per edge this created a two step process to turn on the pump (one edge to receive the on command and one to acquire the mutex), which could be interrupted by other parts of the system meaning a command to turn on the pump might not result in the pump turning on. Turning the pump off had the same problem. Another reason to use two pump templates instead of one is to make it easier to report the status of the pump to the controller. This makes checking the status of the pumps in order to decide what actions to take easier as each pump has a different status variable. This comes at the cost of increased complexity in updating the pumps however as they use separate channels to synchronise on. It also required that careful consideration was given to which pumps get turned on at any given time as the mutex is no longer there to prevent the two from running together.

The sensors simple waiting in the “Standby” state until the controller polls them by pinging the channel. The sensor then either breaks, immediately returning to the standby state until re-polled, or moves to the test state where it reports back to the controller. This process uses probability weights to decide both how often the sensors break and how they respond, in order to allow the system to break and test its response while keeping a reasonable rate of successful responses. The sensors once had urgent states states separating the “Test” and “Standby” states but these were removed in order to get back to “Standby” quicker in the event that the system gets polled again.

The controller handles almost all decisions the system makes. Only the pumps going online and coming back, and sensor behaviour is not its responsibility. The controller waits in its standby state until it shuts down or polls a sensor. The shut down feature is there as it is assumed someone operating the boiler is able to shut the system down manually if they feel it is a fault. The controller has an equal chance of selecting either sensor to poll. Once there it waits five ticks for a response. If no response is received it polls the sensor again. If five polls go unanswered the sensor is assumed to have failed and the system shuts down. If before five polls go unanswered a reply is received from the sensor then system transitions to a temporary state where decisions are mad based upon the status of the pumps.

When the water sensor responds one of three things can occur: a return to “Standby”, a transition to one of two wait states while a decision is made based upon the current pump status. For both low and high water levels there are three possible decisions two of which result in an emergency shut down. For the two that end up in “Standby” if conditions are met the pumps are turned on or off appropriately. In the case where the water is low and no pumps are on and at least one is running a pump is turned on in order to fill the boiler. If the water is high and a pump is running then it is turned off allowing the tank level to reduce.

In the event that the pumps are broken or for high water level broken or off the controller moves to “EmergencyShutdown”. For a high water level where pumps are no longer running but online it is assumed that there is a fault preventing the water from leaving the boiler so the emergency shut down is triggered to prevent damage to the system. A similar situation occurs when the water level is low and a pump is running. As the other pump cannot be turned on to boost the water level it is assumed that there is a fault preventing the water level rising so the system is shut down. This behaviour has been chosen as the boiler and the water in it is not modelled only the sensor responses. These don't provide any context with their readings so the worst has been assumed in most cases in order to prevent damage to the boiler.

Once in the emergency shut down state the model can return the initial state as it assumed that a technician has identified and corrected the problem returning the system to working order. It is also assumed that pumps can brake down. A pump cannot go from on to broken however due to the model requiring the pumps be turned on and off by the controller. Broken or "Offline" pumps can be return to the off state. It is assumed a technician has fixed them.

3. Discuss the safety and security of the boiler control system and the role of model-checking in in testing it.

This model allows the polling of sensors in order to make sure the conditions inside the boiler do not exceed the recommended limits both of water level and steam flow rate. Actions can then be taken either shutting down the system or turning pumps on and off. The sensors themselves are tested to make sure that they are not faulty. If a sensor fails to respond when polled again. If it fails to respond to five polls then the system is shut down as the sensor is assumed to be faulty. This could lead to dangerous conditions inside the boiler which would go unnoticed and unaddressed as the sensor cannot detect them.

The pump behaviour is also monitored to ensure that the system does not take risks when it cannot recover from them. For example if a pump is running and the water level is too low then it might be a fault which puts the system at risk so it shuts down. This feature can be a requirement that is tested and verified by the model checker to ensure that in the event that a risky situation occurs the system shuts down to prevent damage. Other risky scenarios can be safety tested just like this one to make sure the pumps and shut down feature work correctly to avoid damage to the system.

There are two pumps in order to provide some redundancy to the model in the event that one fails the other can fill its role. Only one pump can be active at any time. This can be tested as part of the model to ensure that there is a backup pump available if one goes down. This is not always the case however as both pumps can fail causing the system to shut down. This double failure can also be tested in order verify that the model works correctly. Shutting down if both pumps fail.

This model does not represent the whole system, ignoring many important aspects of a real life boiler such as the heating system, the structural integrity of the system, the actual levels of water and steam in the boiler and other environmental factors. The only information the controller gets about the physical parts of the system is the status of the pumps, whether the steam rate is acceptable or too high and whether the water level is too low, acceptable or too high. More information can be gained from the boiler such as temperature, exact water level and exact steam rate. This information would allow a more accurate model of the boiler to be constructed, testing for more scenarios potentially leading to better fault detection and better up time as situations where faults are assumed in the actual model may not be actual faults but the model has no way to confirm that so shuts down to be safe.

Physical problems with the boiler are not accounted for at all so the model is assuming that the boiler is operating correctly without any physical faults and under normal conditions. This means that it is in an indoor environment, not exposed to the elements with normal temperatures and pressures. The model is designed to assume the worst in several situations where the status of the boiler and pumps does not add up as it does not take into account any external factors and there is not enough information from the inside of the boiler to make an better guess about the conditions inside the boiler. For any situation where the system is exposed to the elements or otherwise put under extra stress such as being physically damaged the model is not valid for it does not take into account these situations in the requirements.

Other situations which invalidate the model are extreme circumstances such as intentional damage done to the system or major disasters such as earthquakes, floods and fires. These have not been included in the model as they cannot be tested for and are unlikely to occur even if they have an impact on the safety of the model. The model is only valid if the system is physically intact and in a normal weather free environment. Not exposed to extreme temperatures or pressures. The heating system is an exception to this. The model cannot detect damage or other external factors so to account for this in some situations where this could have occurred it shuts down and defers to an external technician.

The internals of the system are assumed to be working under similar conditions to the rest of the system. No extreme temperatures or pressures. The liquid in the system should be water. There should be enough of it to flow through the pumps to prevent damage to them. None of this is tested for by the model so it would not be valid under conditions unlike these. It is also assumed that there are no leaks so fluid is retained in the system. This is partially tested for as in the event the water level is low when pumps are running the model assumed damage and shuts down however this is not a comprehensive test as the water level could be rising, there could be a leak or the heat could be too high creating too much steam. As this cannot be directly tested the model is likely not valid in the case of a leak.

The sensors have some tests to ensure they are sending data but there are no tests to make sure the data is valid. The only thing the tests account for is if the sensor responds to requests. If its responses contain incorrect or invalid data there is no way to check and it could cause the system to fail or become damaged. In the event that a sensor is sending bad data the model would not be valid as its decision making is based upon sensor results. However in the event that a sensor become destroyed or is in some way unable to transmit data the model is still valid as this is one of the things that it does test for.

The model has communication between different sections. The controllers, sensors and pumps. Data integrity is important for a system of this kind where bad sensor readouts could lead to disaster if the wrong steps are taken. This model has no system of data verification so despite the fact that no data is being transferred around when implemented data errors could cause problems for the system. Some kind of data id system could be used to verify that data is coming from correct source and it is valid data that can be used properly by the controller.

It is also important to ensure that data is a real command from a valid source. This could mean only accepting data at certain times such as polling a sensor for data rather than being interrupted by it. Getting data only at times where you want it is one way to avoid having unverified data enter the system as there are limited windows for it to enter. There is no scenario modelled for invalid data being entered so the model is not valid for circumstances where bad data enters either through malicious attacks or errors.

4. Critical evaluation of the strengths and weaknesses of TLA+/Pluscal and UPPAAL for the design and verification of computer systems.

UPPAAL is a modelling tool to verify real time systems that can be modelled as real time automata. It also allows synchronising across channels and variables such as integers and doubles amongst other things Behrmann, David, Larsen (2006). It uses clocks to advance time and can have multiple synchronous clocks running at the same time. This makes it good for modelling systems where the progression of time affects the system. This includes many automated systems which run on schedules or any other system with time dependant elements. It can be used to model many complex systems which require many parts to work together concurrently.

One of the benefits of TLA+/Pluscal is that it can be written in Pluscal and converting into TLA+. Pluscal is a programming language which is used for writing algorithms. It can be used to create and test algorithms by translating the Pluscal code into TLA+ and running it through the model checker(Lamport, 2018). This process makes writing and testing algorithms more user friendly than TLA+ alone which while powerful is complex and takes time to learn. Pluscal can help bridge the gap by allowing code similar to python or c to be written and translated allowing a wider range of people to use the model checking and verification features of TLA+.

For a road junction's traffic light system UPPAAL would be the more suitable system as it comes with time as standard. For TLA+/Pluscal an external or custom library would have to be created to allow that kind of functionality. Traffic lights use timers alongside other inputs to determine how long lights should stay a certain colour (Cottingham, no date). These times can be modelled more easily in UPPAAL as clocks a a basic function of the tool kit unlike TLA+/Pluscal making it the better language for this use case. It also has channels to synchronise on so user or sensor inputs can also be easily modelled using UPPAAL for a more comprehensive model with more possible states and conditions that can be tested.

For event driven systems where time does not matter TLA+/Pluscal can be used effectively to create and verify large system models in order to detect bugs or ensure the system logic is correct. For example an entry management system which control who has access to certain parts of a building through door controls could be modelled using TLA+/Pluscal. Its ability to perform mathematical operations with sets make it suitable for dealing with a system like this as all the elements can be broken down into sets which interact with each other through functions to model building access. The people, permissions and areas can all be represented by sets which behave according to the model created. This can then be tested and verified in order to create a mathematical proof that the system's logic is correct.

5. A Critical evaluation of the use of formal methods in Software Engineering for the specification and design of systems.

Formal modelling is a tool that can be used to mathematically verify that a system model meets certain requirements. This can be used to uncover bugs or errors in a designs logic before it is built. These bugs and errors can be fixed before the system goes into production and can save a company ten times as much money than if they found the bug in production (Azevedo, 2018). These bugs are found by creating a precise unambiguous definition of the system which is then modelled and tested. This testing can help find both the bugs and the causes of them by following the trace to the area which broke.

For large or mission critical systems formal modelling can be a vital tool to ensure that a system works properly when implemented. For mission critical systems such as aeroplanes and spacecraft it is vital that their systems work properly or people can get hurt or die. Creating a formal model or several in order to verify the design and make sure it is bug free before it is implemented is very important part of creating systems like these (Chen, 2013). For large systems where peoples lives are not at stake formal modelling is also useful to detect bugs and verify that a system works correctly. Amazon used TLA+/Pluscal to model some of their web services and used it to find bugs in order to roll out software patches without downtime saving their customers money due to their service reminging up and improving their product in the knowledge that the changes will not negatively impact the service (Newcombe et al, 2014).

Creating formal models in not foolproof however. Errors in logic that do not violate invariants or other mistakes such as inaccurate requirements which do not cause testing to fail can provide a designer with proof that their design works. While it it true that the model they tested work this will not translate to the final system due to mistakes in the requirements or other areas (Berry, 2002). This means formal modelling not necessarily the best option for small operations or teams with no experience in the area as the cost of modelling the system may outweigh the benefits of it.

Bibliography

Azevedo, R. (2018) Available at: https://azevedorafaela.com/2018/04/27/what-is-the-cost-of-a-bug/ (Accessed: 17 January 2020)

Behrmann, G. David, A. Larsen, K.G. (2006) Available at: http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf (Accessed: 17 January 2020)

Berry, D.M. (2002) Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.8839&rep=rep1&type=pdf (Accessed: 17 January 2020)

Cottingham, D. (no date) Available at: https://mocktheorytest.com/resources/traffic-lights-uk/ (Accessed: 17 January 2020)

Lamport, L. (2018) Available at: https://lamport.azurewebsites.net/tla/p-manual.pdf (Accessed: 17 January 2020)

Newcome, C. Rath, T. Zhang, F. Munteanu, B. Brooker, M. Deardeuff, M. (2014) Available at: https://lamport.azurewebsites.net/tla/formal-methods-amazon.pdf (Accessed: 17 January 2020)

Chen, Zhe & Gu, Yi & Huang, Zhiqiu & Zheng, Jun & Liu, Chang & Liu, Ziyi. (2013). Model checking aircraft controller software: A case study. Software: Practice and Experience. 45. 10.1002/spe.2242.