

Uppaal model report Dibran Dokter 1047390 & Marnix Lukasse 1047400

We created a starvation-free solution for the readers writer problem. A brief summary of the solution: Whenever a reader or writer wants to perform an action it requests the turnStile semaphore. The first time a reader requests this turnStile (semWait), it will be immediately granted to the reader (semGo). The reader will then immediately release the semaphore again (semSignal). This might seem pointless at first, however this turnStile semaphore actually plays a key role in preventing starvation. To understand this, consider a writer. A writer also request the turnStile semaphore, but releases it only after completing a write. As turnStile is a binary semaphore, this means that no 2 writers can ever be writing simultaneously. Because the semaphore works with a queue mechanism, it prevents starvation. Once a reader or writer requests turnStile, it will be added in the queue. This means that both readers and writers will get the turnStile access eventually.

In this model we have simplified the reading and writing action as a single state and thus it takes no time. Besides this we did not take an simplifying assumptions.

I think the model itself is good, however it is difficult to see how this model can be used by other models. Because of this it may not be very extensible and shareable. Maybe we could use some way of abstracting the parts of the model so it could be easily reused. To check that the model is correct however we have analyzed our model and defined a number of queries to test the following aspects: mutual exclusion in writers and between readers and writers, that there can be more than one reader at a time and that there can not be a deadlock. We defined these queries to prove that there is mutual exclusion between the readers and writers but not between the readers. Uppaal was able to prove these queries, however we were unable to define a query to prove that starvation can not occur. To prove this using uppaal we would need to change the model to check that there is bounded waiting. We were able to prove this by using the simulator however by testing if it was possible to keep reading or writing while there is a reader or writer waiting.

The model only has 5 readers and 2 writers, this is because it would take a long time to verify the queries with more processes. However if we can prove it for 7 processes we also prove it for any number of processes.

In this model we have a state explosion when adding readers and writers, since this greatly increases the number of possible states in the system. However it is not necessary to increase the amount of processes to prove the model as stated above. So we don't have a problem with state explosion in testing our model since we can limit it to a small amount of processes.

The properties claimed by Downey hold, there is mutual exclusion between the readers and writers and there can not be a deadlock. Starvation does not occur in the model. Most of these we could prove using uppaal, except for starvation.

This assignment taught us a lot about correct ways to model and test an algorithm. It also shows the importance of testing an application in this way. However it has also shows the time and effort it takes to correctly model and test an algorithm. It shows why it is not common to formally model and test in this way. We can use this knowledge to decide when we should create a formal model to test an algorithm and we have gained the knowledge on how to build such a model using uppaal.