MoSIS Assignment 4

Team


```
Name       :      Daniëlle Jongstra
Student ID :      20172260
e-mail     :      Danielle.jongstra@student.uantwerpen.be

Name       :      Joanna Kisaakye
Student ID :      20205490
e-mail     :      joanna.kisaakye@student.uantwerpen.be
```


Time spent on this assignment (in hours): --------

The assignment

Petri Net

The petri net is modelled in `Assembly.xml`. The entire net is also shown in fig. 1. There is a simplified petri net in fig. 2, which does not have the time requirements modeled. In this simplified form the general flow of tokens is easier to see, and therefore we'll describe this petri net first.

In the petri net, we start on the left side. We can see the transitions CubeArrives and CylinderArrives that places a new shape of either of them in the queue if the queue is not completely filled yet. The amount of places left in the queue is the amount of tokens in AssemblerQ. As soon as there are at least 2 Cubes and 1 Cylinder in the queue, the transition Assembly becomes enabled, under the extra condition that the assembler is not used already and that there are no remakes queued. The remakes are discussed a little further in the petri net description. If the transition is taken, the token denoting the assembler to be available is taken away, so that another object is not able to enter the assembler. During assembly, there is a token in Assembling denoting that the assembler is in use. After the assembly is done, the transition ToInsp is used. This transition transfers a token back to AssemblyAvailable, as the assembler is no longer in use by taking this transition. The object is then in a queue to enter the inspector. The amount of objects in this queue is kept the same way the queue of the assembler was kept. Transition Insp is taken to move the object from the queue to the inspector. Availability of the inspector is checked the same way it was in the assembler. After this the inspector is left using InspLeave and the object will be in the flagging zone: Inspected. The object can be flagged as either thrashed, accept or remake. If it is flagged as thrashed or accept, there is a token added to respectively ThrashedAmount and AcceptedAmount. These state are counters used for keeping track of the performance. If the object is flagged as a remake, there is a token added to the counter RemakeAmount which is also used for tracking the performance. Another thing that is done when an object is flagged as remake is that is takes a token from RemakeNotQueued. This place has 10 tokens if there are no remakes waiting for the assembler. For each remake waiting a token is taken away. There is also a token in RemakeQ, which is the object to remake being in the queue. The remake can enter the assembler by taking the transition AssemblyRemake. After this transition is taken, the object will follow the same route as described before. If there is a remake, the assembler can not make a new object from 2 Cubes and a Cylinder. The transition that would transfer a new object to the assembler requires 10 tokens in RemakeNotQueued. Because a token is taken away every time a new remake is flagged, the transitions Assembly and AssemblyRemake can not be enabled at the same time.

The complexity in fig. 1 comes from adding the time and the necessary checks that came with that. Most of the places related to the time are on the bottom half of the petri net. The time works by having a token available in Time_NextAvailable to go to the next time step. DoNextStep then distributes a token to each of the 3 main processes (shapes, assembler and inspector) to denote that they can still do something during that time slice. If a process is unable to do something, chooses to skip acting the time slice or does an action, the token denoting an action is available is transferred to a place denoting the process has been executed during the time slice. Extra logic is added to check so that a process can only skip processing if the process before it already attempted to process. This way, a process that was not able to become active without another process already having executed is still able to execute after this process did their step. The assembler and inspector have some new places to check if they are able to execute during the time slice. They can only skip if these places denote that they are not able to execute. As soon as all the processes have either skipped or executed, the transition NextStepAvailable becomes available. This takes the tokens from the executed places and transfers one to the Time_NextAvailable place. This means the time logic can start from the beginning.

Figure 1: The complete petri net

Figure 2: A simplified petri net, not keeping time into account

Simulation 1

Simulation 1 is the case of "The Assembler finished an assembly at the exact same timestep the Inspector marks an assembly for reassembly". The simulation is shown in fig. 3 till fig. 7. In the first step, it can be seen that both the inspector and the assembler can finish.

Figure 3: Step 1 in simulating case 1

Step 2 shows that within the same time step the object leaves the inspector, it can be flagged as a remake.

Figure 4: Step 2 in simulating case 1

Step 3 shows that an object leaves the assembler. The other object is in the remake queue.

Figure 5: Step 3 in simulating case 1

Step 4 shows that when the assembly object is moved to the inspector queue, the transition for the remake object to the assembly procedure becomes enabled.

Figure 6: Step 4 in simulating case 1

Step 5 shows that after entering the assemble procedure, the object can move into the actual assembly. It is possible that the object is in the assembler for the first time step during the same time slice. This however depends on the leaving object. If the leaving object used the assembler during the time slice, the new object will only gain being in the assembler for 1 time unit in the next time slice. Else it is possible for the new object to be in the assembler for one time unit during the current time slice.

Figure 7: Step 5 in simulating case 1

Simulation 2

Simulation 2 is the case of "There is an assembly waiting for reassembly (i.e., it is to be fixed) and at that exact time the third component of an assembly arrives". This can be seen in fig. 8 and fig. 9. The first step its blue circles show that there is an assembly for remake in the remake queue and that it is not yet possible to assemble a new object. The pink circle shows shows that in this time step, the last needed shape can arrive.

Figure 8: Step 1 in simulating case 2

The pink circle in the second step shows that the last shape has arrived, but that the transition to the assembler for the new object is not enabled. The blue circle shows that the transition for the remake is enabled.

Figure 9: Step 2 in simulating case 2

Reachability

Invariants

- **M(AssemAble) + M(AssemblyActive) + M(AssemblyAvailable) = 1**: This invariant was not expected, but does make sense. AssemblyAvailable denotes that no object has tried to enter the assemble procedure. When an object does try, the single token is taken from AssemblyAvailable and moved to AssemAble. This means that the object is trying to get in the assembler. If this happens, the token is taken from AssemAble and moved to AssemblyActive. When the assembler is done, the token is taken from AssemblyActive and moved to AssemblyAvailable. This means that there is never a token in all three places at the same time.
- **M(AssemAble) + M(AssemblyActive) + M(AssemUnable) = 1**: This was definitely an unexpected invariant. The expected invariant consisted of AssemAble and AssemUnable being 1, as the assemble procedure is either available or not. That AssemblyActive is included in this, is because the token does not move immediately from AssemUnable to AssemAble. It first goes to AssemblyActive to be transferred back to AssemAble when the assembler frees up.
- **M(AssemblerQ) + M(CubesInQ) + M(CylindersInQ) = 10**: This invariant was expected. That the available assembly queue and the amount of the cubes in the queue and the amount of cylinders queued are a total of 10.
- **M(InspAble) + M(InspectinAvailable) + M(Inspecting) = 1**: the same reasoning goes for this as for the the equal assembly invariant (first on the list).
- **M(InspQ) + M(InspQAvailable) = 10**: This invariant was expected, as the token moves from the available queue to the actual queue every time an object arrives.
- **M(InspAble) + M(Inspecting) + M(InspUnable) = 1**: For this invariant, the same goes as for the related assembler invariant (bullet-point 2)
- **M(RemakeNotQueued) + M(RemakeQ) = 10**: same goes as for the inspector queue.
- **M(Time_Assembly) + M(Time_AssemblyExecuted) + M(Time_NextAvailable) = 1**: This invariant was expected, because the Time_Assembly denotes that the assembler can still do a step in that time slice, while Time_AssemblyExecuted has a token when the assembler has executed in that time slice. Time_NextAvailable contains a token if all processes had a token which can be taken away, so Time_AssemblyExecuted does not have the token anymore as soon as Time_NextAvailable has one. It would be bad if there were multiple tokens in these three places combined.
- **M(Time_InspectingExecuted) + M(Time_Inspection) + M(Time_NextAvailable) = 1**: same goes as for the Time_Assembly equation above
- **M(Time_NextAvailable) + M(Time_Shape) + M(Time_ShapeExecuted) = 1**: same goes as for the Time_Assembly equation above
- **M(AssemblerQ) + M(TotalInQAssem) = 10**: same goes as for the inspector queue.

The analysis of these invariants is in `invariants.html`. The invariants do no change by adding a stock. The invariants can be changed by adding a single place. In the example that generated `invariantChange.html` a place InvarChange was added. It has 200 tokens at the start of a simulation. Every time a cylinder arrives, a token is taken, and every time a cylinder an two cubes are moved to the assembler, a token is added. This makes that the cylinder queue and the InvarChange always have 200 tokens cumulative. This adds the last invariant on the list, **M(CylindersInQ) + M(InvarChanger) = 200**.