Technical University of Denmark

02246 Model Checking

Mandatory Assignment Part 1: Discrete Modelling and Verification

Authors:

Andreas Hallberg Kjeldsen
s092638@student.dtu.dk

Morten Chabert Eskesen
s133304@student.dtu.dk

October 21, 2013

Part A: Introductory Problems

Practical Problems

1.

(a)

$client_1$: $AG(state_1 = 1 \Rightarrow \neg(\neg job_1 = 1 \wedge \neg job_2 = 1))$
$client_2$: $AG(state_2 = 1 \Rightarrow \neg(\neg job_1 = 2 \wedge \neg job_2 = 2))$

(b)

$P >= 1[Gstate1 = 1 => (job1 = 1)|(job2 = 1)]$ - Verified.
$P >= 1[Gstate2 = 1 => (job1 = 2)|(job2 = 2)]$ - Verified.

(c)

$client_1$: $AG(state_1 = 1 \Rightarrow job_1 = true)$
$client_2$: $AG(state_2 = 1 \Rightarrow job_2 = true)$
We require that whenever $state_1 = 1$ then $job_1 = true$ because there should be a job waiting in the queue when the $state_1 = 1$. The same goes for $client_2$.

(d)

$P >= 1[Gstate1 = 1 => job1 = true]$ - Verified.
$P >= 1[Gstate2 = 1 => job2 = true]$ - Verified.

2.

(a)

In order for the *Scheduler* to cope with an extra client we first add an extra module called $client_3$ with the same commands as the two other clients with the names of the commands corresponding to $client_3$. We changed the finite range, which job_1 and job_2 can take their value over to $0 \ldots 3$. This does not increase the length of the queue because there is still only two jobs allowed in the queue (job_1 and job_2). We also added commands create3, serve3 and finish3 and only changed the values according to the number of $client_3$.

(b)

In the new model there are 214 reachable states.

(c)

A client cannot create a job when the queue is full. This is because all the modules synchronize over all action names that appear syntactically in the modules. The commands create1, create2 and create3 are also in scheduler with a guard that specifies that the $job_2 = 0$ for creation of a job to be possible, and $job_2 = 0$ is only true if the queue is empty.

(d)

Yes the properties previously verified still hold in the new model. They do because the clients' states will only be 1 if their job is in the scheduler since the modules are synchronized.

3.

(a)

In order to modify the queue to have a length of three we add another job to the queue called job_3 which will hold the third job of the queue. We also changed the create commands to have the guard $job_3 = 0$ because now this is the last job in the queue, so when it is 0 there is a place for one more job. Furthermore we added another method for shifting the queue when there is an empty slot. The old command stays in place, but there is now another command with the guard $job_2 = 0$ & $job_3 > 0$ that shifts job_3 to job_2 so it is moved up in the queue. Since the commands have no action names the commands can always occur *independently* of what any other modules in the systems are doing - just so long as its guard is true.

(b)

In the new model there are 1459 reachable states.

(c)

The properties previously verified do not hold in the new model, because the queue is now of length 3. Which means that a job created by a client could be in scheduled as the last job, i.e. in job_3. Example: this would cause (for $client_1$) to have $state_1 = 1$ while $job_3 = 1$ because the job is at the end of the queue.

(d)

???????????????????????????

4.

(a)

$AG\ \phi$ specifies that from all the paths from this state ϕ should hold. Whereas property ϕ should only hold in that state.

(b)

The semantics are different in the version of PRISM we use. If the property ϕ should hold in all reachable states it should be written $AG\ \phi$. Because if only ϕ has been written as the property - this version of PRISM will only check if the property ϕ holds in the *initial* state.

(c)

(d)

Theoretical Problems

1.

(a)

(b)

2.

(a)

(b)

(c)

(d)

(e)

3.

(a)

(b)

(c)

(d)

4.

Part B: Intermediate Problems

Practical Problems

1.

(a)

In order to model a round-robin scheduler from the SRT scheduler we introduce some variables to the *Scheduler* module.

turn : [0..2] init 0; // Who is next?
job1time : bool init false; // Has job1 used up its time?
job2time : bool init false; // Has job2 used up its time?

The turn variable specifies which job is next to be served. When turn=1 the next job to be served is job1.
The jobXtime variable specifies if jobX has used up all its serving time in the scheduler. So if jobXtime is true the next job should be served whereas if jobXtime is false this job should be served soon.
We also introduce some new commands with no action names in the module. These commands can always occur independently of what any other modules in the system are doing. Its guard just has to be true.

[] job1=true & job1time=true & turn=1 ⇒ (job1time′=false);
[] job2=true & job2time=true & turn=2 ⇒ (job2time′=false);

These commands are used to switch between the jobs. If job1 has used up all its time in the scheduler but it is now job1s turn in the queue job1time should not be true. It should then be modified to false so it can be served by the scheduler. The same goes for job2.
Furthermore we have also modified the serve commands.

[serve1] job1=true & job2=false ⇒ true;
[serve2] job1=false & job2=true ⇒ true;
[serve1] job1=true & job2=true & turn=1 & job1time=false ⇒ (job1time′=true) & (turn′=2);
[serve2] job1=true & job2=true & turn=2 & job2time=false ⇒ (job2time′=true) & (turn′=1);

It serves the jobs if there is no other job in the queue. That should not be modified. However it should only serve a job if it is the jobs turn to be served and the job has not used up its time being served. This is in the guard specified as turn has to correspond with the clients number and its jobtime has to be false.

(b)

We have made some additions to the *create* and *finish* commands. The create command now has two different kinds for each client. One is the same as the SRT scheduler. The other command has another guard - turn=0. This command is only used if there has not been any jobs scheduled in the round-robin scheduler before the creation of the new job. When turn=0 there has not been scheduled any jobs yet because turn is initialized as 0 and then turn is only altered between 1 and 2.

$$
\begin{gathered}
\text{[create1] job1=false \& turn¿0} \Rightarrow \text{(job1}'\text{=true);}\\
\text{[create1] job1=false \& turn=0} \Rightarrow \text{(job1}'\text{=true) \& (turn}'\text{=1);}\\
\text{[create2] job2=false \& turn¿0} \Rightarrow \text{(job2}'\text{=true);}\\
\text{[create2] job2=false \& turn=0} \Rightarrow \text{(job2}'\text{=true) \& (turn}'\text{=2);}\\
\dots\\
\text{[finish1] job1=true} \Rightarrow \text{(job1}'\text{=false) \& (job1time}'\text{=false);}\\
\text{[finish2] job2=true} \Rightarrow \text{(job2}'\text{=false) \& (job2time}'\text{=false);}
\end{gathered}
$$

The finish commands has been altered to also set the variable jobXtime to false. Since the job is now finished its time in the queue should not be specified as used up. It should therefore be sat as false so it does not affect the next job created by the same client.

(c)

In the new model there are 277 reachable states.

(d)

$P >= 1[Gstate1 = 1 => job1 = true]$ - Verified.
$P >= 1[Gstate2 = 1 => job2 = true]$ - Verified.

2.

(a)

The *Client* module has been modified with an additional variable called *priority* which can be either 1 or 2. When the priority is 2 the job has a higher priority than a job with priority 1. The create commands have been altered to 10 new ones so the client is able to create jobs with both priority 1 and priority 2.

$$
\begin{gathered}
\text{priority1 : [1..2] init 1; // Priority of the job}\\
\text{[create1] state1=0} \Rightarrow \text{(state1}'\text{=1) \& (task1}'\text{=1) \& (priority1}'\text{=1);}\\
\dots\\
\text{[create1] state1=0} \Rightarrow \text{(state1}'\text{=1) \& (task1}'\text{=1) \& (priority1}'\text{=2);}\\
\dots
\end{gathered}
$$

(b)

The *Scheduler* module is modified so that when a new job is created and it has a higher priority than a job in the queue it moves ahead of that job. This is done in the module by creating some new create commands in the Scheduler module.

[create1] job2=0 $\Rightarrow$ (job2′=1);
[create2] job2=0 $\Rightarrow$ (job2′=2);
[create1] job2=0 & priority1=2 & priority2=1 $\Rightarrow$ (job2′=job1) & (job1′=1);
[create2] job2=0 & priority2=2 & priority1=1 $\Rightarrow$ (job2′=job1) & (job1′=2);

When a new job is created by any of the clients and it has a priority of 2 and the existing job in the queue has a priority of 1 then the new job moves ahead in the queue and the existing job moves back. If the priorities of the jobs are the same no changes should be made and likewise when the existing job in the queue has a priority of 2 and the newly added job has a priority of 1 then no changes should be made.

(c)

In the new model there 360 reachable states.

(d)

$P >= 1[Gstate1 = 1 => (job1 = 1)|(job2 = 1)]$ - Verified.
$P >= 1[Gstate2 = 1 => (job1 = 2)|(job2 = 2)]$ - Verified.

Theoretical Problems

1.

(a)

(b)

(c)

(d)

(e)

(f)

2.

(a)

(b)

(c)

(d)

Part C: Advanced Problems

Practical Problems

We have modified our round-robin scheduler to include priorities. In our specification the scheduler choses the job with the highest priority and it serves it till its finished unless the other job has waited 5 turns. This avoid starvation but it could also keep an important job in the queue while serving a less important job. If the priority is equal the scheduler switches between the jobs in a regular round-robin fashion.

No.	CTL	PRISM notation	Verified
1.	$AG(state_1 = 1 \Rightarrow job_1 = true)$	$P >= 1[Gstate1 = 1 => job1 = true]$	✓
2.	$AG(state_2 = 1 \Rightarrow job_2 = true)$	$P >= 1[Gstate2 = 1 => job2 = true]$	✓
3.	$task_1 > 0 \Rightarrow AFtask_1 = 0$	$task1 > 0 => P >= 1$ [F task1=0]	✓
4.	$task_2 > 0 \Rightarrow AFtask_2 = 0$	$task2 > 0 => P >= 1$ [F task2=0]	✓