02246 Mandatory Assignment
A02 - Model Checking Algorithms[*]

To be submitted on DTU Learn - see deadline on DTU Learn

You are encouraged to work in groups, but you must clearly identify the contributions of each group member, and you will be jointly responsible for the finished report. Register your group on DTU Learn before submitting as group submission.

Answers to all parts should be typed up using LaTeX and submitted electronically as a PDF report using the provided template. Drawings and formulae may be handwritten and scanned. More detailed instructions as to the style of answer we expect for each part are included below.

Some tasks require to upload files.

[*]Thanks to Michael Smith (original author), and Lijun Zhang, Kebin Zeng and Flemming Nielson and Alberto Lluch Lafuente (contributors)

A02 - Model Checking Algorithms

A02P: Practical Problems

A02P.1 So far we have looked at just two scheduling disciplines — FCFS and SRT. Another type of scheduler that is very widely used is the *round-robin* scheduler. Rather than running each job to completion before moving onto the next, as in the FCFS scheduler, or making the decision of which job to serve next at each time step, as in the SRT scheduler, a round-robin scheduler executes each task for just one time unit before moving onto the next, cycling through them.

a) Think about how you would model such a scheduler in PRISM, for a scenario with two clients. Using the SRT scheduler as a starting point, modify the *serve* commands, so that the scheduler executes jobs in a round-robin fashion.

You may want to approach this by first assuming that there are always two waiting jobs (i.e. job_1 and job_2 are always true), and figuring out how to cycle between them. *Hint: you will need to add some extra state to the Scheduler module.* You can then modify your solution, so that it behaves correctly when there are fewer than two jobs waiting. *Hint: look at how we shifted the queue in the FCFS scheduler.*

Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

b) Make any changes that are needed to the *create* and *finish* commands, to complete the module. Explain your changes.

Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

UPLOAD REQUIRED: the new prism model `A02P.1.b.prism`.

c) How many states does your model have? Provide a screenshot.

Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

d) Modify the properties that you have previously verified for the FCFS and SRT schedulers, and verify them for your model. Provide a screenshot showing the results.

Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

UPLOAD REQUIRED: the new prism properties `A02P.1.d.props`.

e) If a property fails to hold, check whether this was due to a mistake in your model, or is a real problem with the round-robin scheduler. In the case of the former, explain what the mistake was, and fix it in your model. In case of the latter, explain what the problem is.

Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

A02P.2 In many real schedulers, we need to take into account that tasks have different *priorities*. For example, operating system processes for tasks such as memory management and hardware control need to be given priority over user level processes such as a word processor or web browser. Your challenge is to extend the FCFS server to handle tasks with two different priority levels.

a) Modify the *Client* module so that it can create tasks with two different priority levels. You need to decide how to encode the priorities — do you use an additional variable, or do you encode it in the actions? (note: there is not only one correct answer here). Explain the main ide of your new model.

   Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

b) Modify the *Scheduler* module so that when a new job arrives, it moves ahead of any lower priority jobs that are in the queue — this may involve pre-empting (interrupting the execution of) the currently running job so that the higher priority job can run. Note that how you do this will depend on the choices you made in modifying the *Client* module. Explain your solution.

   Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

   UPLOAD REQUIRED: the new prism model `A02P.2.b.prism`.

c) How many states does your model have? Provide a screenshot.

   Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

d) Modify the properties that you have previously verified for the FCFS schedulers, and verify them for your model. Provide a screenshot showing the results.

   Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

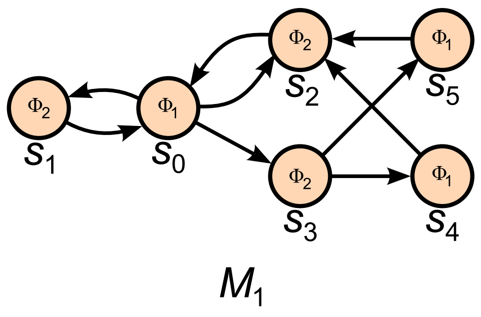
   UPLOAD REQUIRED: the new prism properties `A02P.2.d.props`.

e) If a property fails to hold, check whether this was due to a mistake in your model, or is a real problem with the priority scheduler. In the case of the former, explain what the mistake was, and fix it in your model. In case of the latter, explain what the problem is.

   Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

   UPLOAD REQUIRED: the new prism model `A02P.2.e.prism`.

A02: Theoretical Problems



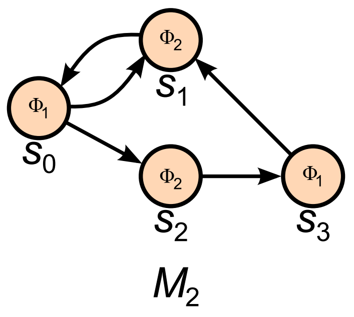


Figure 1: Two transition systems, M_1 and M_2

A02T.1 Consider the CTL formula $\Phi = \neg(EX\ \Phi_1) \wedge (AF\ \Phi_2)$, where Φ_1 and Φ_2 are atomic propositions. We will apply the model checking algorithm, described in Section 6.4 of the book.

a) Convert Φ into $\exists$-normal form, and draw the abstract syntax tree of the resulting formula, which we will call Φ'.

Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

b) Compute the set of satisfying states $Sat(\Phi) = Sat(\Phi')$ for the transition system M_1 from Figure 1, explaining how you do this, including how you computed the satisfaction set of each sub-formula.

Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

A02T.2 Design algorithms to the compute below satisfaction sets directly (without converting them into ECTL). For each case, sketch the algorithm as pseudo-code (as in the slides) and argue for its correctness.

HINT: Start by having a look at the algorithms seen in class for the existential variants of the below formulas. Ask yourself what you would change to make them work for universal quantification ($\forall$) instead of existential quantification ($\exists$). When you provide your answer, you will likely want to highlight (with some color) the difference between both cases.

a) $sat(\forall(\mathsf{X}\phi))$

Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

b) $sat(\forall(\phi_1 \mathsf{U} \phi_2))$

Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

c) $sat(\forall\Box\phi)$

Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

A02T.3

1. Consider the two transition systems in Figure 1: $M_1 = (S_1, \rightarrow_1, \{ s_0 \}, AP, L_1)$ and $M_2 = (S_2, \rightarrow_2, \{ s_0 \}, AP, L_2)$, where $AP = \{ \Phi_1, \Phi_2 \}$.

   (a) Write down the coarsest (i.e. the largest) bisimulation relation you can find between the states of the two structures: $R_{12} \subseteq S_1 \times S_2$.
   Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

   (b) Argue why R_{12} is a bisimulation, and argue why it is maximal.
   Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

   (c) Write down the coarsest bisimulation relation you can find between the states of M_1: $R_{11} \subseteq S_1 \times S_1$. For conciseness, provide the relation as a partition of the states (i.e. as the set of equivalence classes), instead of enumerating all pairs in the relation.
   Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

   (d) Argue why R_{11} is a bisimulation, and argue why it is maximal.
   Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

   (e) Construct the bisimulation quotient $M_1/\sim$ of M_1.
   Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.

   (f) Is the quotient bisimilar to M_2? If so, what is the largest bisimulation relation between them?
   Provide your answer here. Leave the special color (blue). Figures, tables, code snippets can be placed somewhere else but they need to be referred here.