02246 Mandatory Assignment
Part 1: Discrete Modelling and Verification

Flemming Nielson, Michael Smith, Lijun Zhang, Kebin Zeng

Set: September 10, 2013

Due: 08:00, October 21, 2013

To be handed in electronically through CampusNet.

This assignment is concerned with Part I of the course — discrete modelling and verification. The problems are divided into three parts, with increasing levels of difficulty. All the problems in the first two parts are compulsory, however you may choose between focussing on the practical or the theoretical aspects of the course for the more advanced problems in Part C.

You are expected to spend between 20 and 25 hours on this assignment. Of this, you should expect to spend roughly 6–8 hours on Part A, 9–11 hours on Part B, and 5–6 hours on Part C. You are encouraged to work in groups of two (or at most three) people, but you must clearly identify the contributions of each group member, and you will be jointly responsible for the finished report.

Answers to all parts should be typed up and submitted electronically as a report, although 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.

Part A: Introductory Problems

You are required to answer all *of the problems in Parts A1 and A2. These are short problems, aimed at reinforcing your basic understanding of the course material. Answers should be self-contained, but concise, and you should clearly specify which question you are answering.*

Part A1: Practical Problems

1. For the FCFS scheduler, we would like to verify that whenever a client has an active job, the scheduler has that job somewhere in its queue. For example, in the case of the first client, we require that whenever $state_1 = 1$, then either $job_1 = 1$ or $job_2 = 1$.

   (a) Express this as two CTL properties — one for each client.

   (b) Use PRISM to verify whether these properties hold of the FCFS scheduler model.

   (c) Write down two similar properties for the SRT scheduler, explaining your construction.

   (d) Verify whether they hold of the model.

2. Add another client to the PRISM model of the FCFS scheduler. You will need to modify the *Scheduler* module to cope with the extra client, but for now do not increase the length of the queue.

   (a) Explain the changes that you made to the model, and argue why they satisfy the above instructions.

   (b) How many reachable states are in the new model?

   (c) What will happen if the queue is full when a client attempts to create a job?

   (d) Do the properties you have previously verified still hold of the model? If not, why not?

3. Now modify the *Scheduler* module so that the queue is of length three.

   (a) Explain the changes that you made to the model, and argue why they are correct.

   (b) How many reachable states are in the new model?

   (c) Do the properties now hold of the model? If not, why not?

   (d) Can you give an upper bound on the number of reachable states for a model with n clients, and a queue of length m?

4. Consider the CTL properties Φ and $AG\ \Phi$, where Φ is an atomic property.

   (a) What are their semantics, and how do they differ?

   (b) Are their semantics different in the version of PRISM that you use?

   (c) How would you solve the classical model checking problem $\mathcal{M}, s \models \Phi$ as a problem of the form $\forall s' : \mathcal{M}, s' \models \Phi'$ (where you can define Φ' and may assume that all states in the model are reachable)?

   (d) How would you solve the model checking problem $\forall s' : \mathcal{M}, s' \models \Phi$ as a problem of the form $\mathcal{M}, s \models \Phi'$ (where you can define Φ' and may assume that all states in the model are reachable)?

Part A2: Theoretical Problems

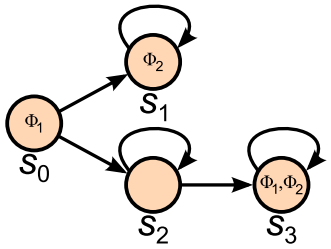

Figure 1: A transition system

1. Consider the transition system, shown graphically in Figure 1. The states are represented by circles, whose names are shown beneath them, and whose labels are shown inside them. The initial state is s_0.

(a) Write down this transition system[1] formally, as a tuple $(S, \rightarrow, S_0, AP, L)$.

(b) By directly reasoning with the semantics of CTL, determine whether the following properties hold of the state s_0:

i. $AF\ \Phi_2$.

ii. $AX\ \Phi_2$.

iii. $EF\ \Phi_1$.

iv. $A[\Phi_1\ U\ \Phi_2]$.

2. For each of the following pairs of CTL formulae, determine whether (a) they are equivalent, (b) one implies the other, or (c) neither implies the other. Explain your reasoning.

(a) $EX\ EF\ \Phi$ and $EF\ EX\ \Phi$.

(b) $AX\ AF\ \Phi$ and $AF\ AX\ \Phi$.

(c) $AG\ EF\ \Phi$ and $EF\ AG\ \Phi$.

(d) $AG\ (\Phi_1 \wedge \Phi_2)$ and $(AG\ \Phi_1) \wedge (AG\ \Phi_2)$.

(e) $EF\ (\Phi_1 \wedge \Phi_2)$ and $(EF\ \Phi_1) \wedge (EF\ \Phi_2)$.

3. Write down a CTL* formula for each of the following properties, which are described in natural language. In each case, argue whether or not the property can also be expressed in CTL.

(a) There is a path on which Φ holds infinitely often.

(b) For all paths, Φ_1 holds along the path until Φ_2 holds of a state and Φ_3 holds of the state that immediately follows.

(c) There is a path on which either Φ_1 eventually holds or Φ_2 eventually holds.

(d) For all paths, either Φ_1 always holds or Φ_2 always holds.

4. Encode the transition system in Figure 1 as a PRISM module, using a variable s, such that $0 \leq s \leq 3$, to represent the state. Define Φ_1 and Φ_2 as PRISM labels. *You should try using PRISM to model check the CTL formulae in question 1(b), and make sure that it agrees with your answers (you don't need to include this in your answer).*

Part B: Intermediate Problems

You are required to answer all *of the problems in Parts B1 and B2. These are more involved than the problems in Part A, and require a more detailed understanding of the course material. Your answers should be submitted as a self-contained report — rather than providing a list of answers, you should present a write-up that is readable, explanatory, and addresses all of the questions posed.*

[1] Here, we take a slight departure from the course notes, in considering a transition system *without* actions. In this case, a transition system is a 5-tuple $(S, \rightarrow, S_0, AP, L)$, where $\rightarrow \subseteq S \times S$ is a *total* relation (i.e. every state has at least one transition out of it).

Part B1: Practical Problems

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.*

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

   (c) How many states does your model have?

   (d) Modify the properties that you have previously verified for the FCFS and SRT schedulers, and verify them for your model.

   (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.

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).

   (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.

   (c) How many states does your model have?

   (d) Modify the properties that you have previously verified for the FCFS schedulers, and verify them for your model.

(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.

Part B2: Theoretical Problems

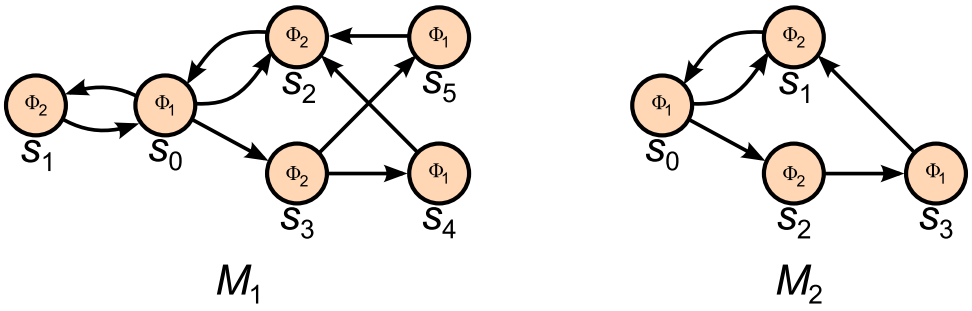

Figure 2: Two transition systems, M_1 and M_2

1. Consider the two transition systems in Figure 2: $M_1 = (S_1, \rightarrow_1, \{ s_0 \}, AP, L_1)$ 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: $R_{12} \subseteq S_1 \times S_2$.

   (b) Show that R_{12} is a bisimulation, and argue why it is maximal.

   (c) Write down the coarsest bisimulation relation you can find between the states of M_1: $R_{11} \subseteq S_1 \times S_1$.

   (d) Show that R_{11} is a bisimulation, and argue why it is maximal.

   (e) Construct the bisimulation quotient $M_1/\sim$ of M_1.

   (f) Is this bisimilar to M_2? If so, what is the largest bisimulation relation between them?

2. 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 notes.

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

   (b) For each sub-term Ψ in Φ' (including Φ' itself), define $Sat(\Psi)$ — the set of states that satisfies the formula — in terms of the sets of states that satisfy each of the sub-terms of Ψ. You may assume that $Sat(\Phi_1)$ and $Sat(\Phi_2)$ are already defined. Note that these definitions should be general, with respect to an arbitrary transition system $(S, \rightarrow, S_0, AP, L)$.

   (c) Find a sub-term Ψ of Φ' whose set of satisfying states is defined recursively. We compute the set as a *fixed point* of this equation. Explain the difference between a *least* and a *greatest* fixed point in general, and how we can compute each. In the case of your equation, which fixed point is correct? What would the other fixed point correspond to?

   (d) Compute the set of satisfying states $Sat(\Phi) = Sat(\Phi')$ for the transition system M_1 from Figure 2, explaining how you do this.

Part C: Advanced Problems

You are required to answer the problems in just one *of the following parts* — either *Part C1* or *Part C2. These are more open ended problems, allowing you to explore some of the concepts in the course in more depth. Your answers should be submitted as a report, as for Part B.*

Part C1: Practical Problems

Modify your round-robin scheduler to include priorities. You will need to make a decision as to how the priorities affect the scheduling of jobs. Is it possible to build a scheduling discipline with priorities that avoids starvation? What implications does this have? Specify some additional properties of your scheduler in CTL, explaining what they mean in natural language, and use PRISM to model check them — if a property fails to hold, give a counter-example and discuss whether you could modify your model to satisfy it.

Part C2: Theoretical Problems

As a logic, CTL has the problem that it considers *all* possible paths in a model, including infinite paths where a transition that is possible infinitely often (due to non-determinism) is never taken. Formally define a notion of a *fair* path that excludes such behaviours, and explain your choice of definition. Is it possible to encode this notion of fairness as a property in CTL*? What about CTL? Construct a variant of CTL whose operators have a fair semantics, according to your notion of fairness.