Solutions to exercises for the Part II course
Temporal Logic and Model Checking

Q1

This question concerns the software example `DIV`.

```
0:  R:=X;
1:  Q:=0;
2:  WHILE Y≤R DO
3:  (R:=R-Y;
4:   Q:=Q+1)
5:
```

Write down a total relation $\hat{R}_{\texttt{DIV}}$ that agrees with the relation $R_{\texttt{DIV}}$ given in the slides where $R_{\texttt{DIV}}$ is defined. If $R_{\texttt{DIV}}$ specifies no successor to state s then what is the successor to s specified by $\hat{R}_{\texttt{DIV}}$?

Solution

$$S_{\texttt{DIV}} = [0..5] \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \qquad \text{(where } [m..n] = \{m, m{+}1, \ldots, n\}\text{)}$$

$$\begin{array}{lll}
\multicolumn{3}{l}{\hat{R}_{\texttt{DIV}}\ (pc, x, y, r, q)\ (pc', x', y', r', q') =} \\
\quad (pc = 0) & \Rightarrow ((pc', x', y', r', q') = (1, x, y, x, q)) & \wedge \\
\quad (pc = 1) & \Rightarrow ((pc', x', y', r', q') = (2, x, y, r, 0)) & \wedge \\
\quad (pc = 2) & \Rightarrow ((pc', x', y', r', q') = & \\
 & \qquad \textit{if } y{\leq}r \textit{ then } (3, x, y, r, q) \textit{ else } (5, x, y, r, q)) & \wedge \\
\quad (pc = 3) & \Rightarrow ((pc', x', y', r', q') = (4, x, y, (r{-}y), q)) & \wedge \\
\quad (pc = 4) & \Rightarrow ((pc', x', y', r', q') = (2, x, y, r, (q{+}1)) & \wedge \\
\quad (pc \notin [0..4]) & \Rightarrow ((pc', x', y', r', q') = (pc, x, y, r, q) &
\end{array}$$

If $R_{\texttt{DIV}}$ specifies no successor to state s then s is the successor to s specified by $\hat{R}_{\texttt{DIV}}$.

Q2

This question concerns the software example `DIV`.

(a) Write down an atomic property that expresses “when `DIV` has halted $r < q$”.

(b) Compute the set of states $\{(pc, x, y, r, q) \mid R^*_{\texttt{DIV}}\ (0, 7, 2, r_0, q_0)\ (pc, x, y, r, q)\}$.

Does the atomic property (a) hold of all states in the set computed for (b)?

Solution

(a)

$$\texttt{DivHaltLess}\ (pc, x, y, r, q) \;=\; (pc = 5) \Rightarrow r < q$$

(b)

$$\begin{aligned}
&\{(pc, x, y, r, q) \mid R^*_{\texttt{DIV}}\ (0, 7, 2, r_0, q_0)\ (pc, x, y, r, q)\} \\
&\;= \{(0, 7, 2, r_0, q_0), (1, 7, 2, 7, q_0), (2, 7, 2, 7, 0), (3, 7, 2, 7, 0), (4, 7, 2, 5, 0), \\
&\qquad (2, 7, 2, 5, 1), (3, 7, 2, 5, 1), (4, 7, 2, 3, 1), (2, 7, 2, 3, 2), (3, 7, 2, 3, 2), \\
&\qquad (4, 7, 2, 1, 2), (2, 7, 2, 1, 3), (5, 7, 2, 1, 3)\}
\end{aligned}$$

Q3

Modify the definition of $\mathsf{Path}\ R\ s\ \pi$ given in the slides to work when the transition relation R is represented as a set of pairs of states, $R \subseteq S \times S$, rather than as a function $R : S \to S \to \mathbb{B}$ (as is done in the slides).

Solution

$$\mathsf{Path}\ R\ s\ \pi \;=\; (\pi(0) = s)\ \wedge\ \forall i.\ (\pi(i), \pi(i{+}1)) \in R$$

Q4

Define a model M and atomic property ϕ such that $M \models \mathbf{GA}\phi$ represents the property: if `DIV` is run in a state satisfying atomic property P and if it terminates, then in the state in which it terminates atomic property Q holds (this is the partial correctness $\{P\}\texttt{DIV}\{Q\}$ in Hoare logic notation).

Can you represent the property that `DIV` always terminates when started in a state satisfying P in the form $M \models \mathbf{GA}\phi$, for suitable ϕ? Justify your answer.

Solution

Note: this question is rather poorly worded in that it might be unclear exactly what "atomic property P" and "atomic property Q" are properties of. In the solution below, I take P and Q to be properties of the whole state (pc, x, y, r, c). However, I think it makes more sense, given the analogy with Hoare triples $\{P\}C\{Q\}$, for P and Q just to be predicates on the 'data part' of the state, namely properties of (x, y, r, c), and to ignore the program counter pc. A solution based on such an interpretation of P and Q would be fine.

$$\begin{aligned}
\mathsf{Reachable}\ M &= \{s' \mid \exists s \in S_0.\ R^*\ s\ s'\} \text{ and} \\
M \models \mathbf{AG}\,\phi &\Leftrightarrow \mathsf{Reachable}\ M \subseteq \{s' \mid \phi(s')\} \\
&\Leftrightarrow \{s' \mid \exists s \in S_0.\ R^*\ s\ s'\} \subseteq \{s' \mid \phi(s')\} \\
&\Leftrightarrow \forall s'.\ s' \in \mathsf{Reachable}\ M \Rightarrow \phi(s')
\end{aligned}$$

Take $M = (S_{\texttt{DIV}}, \hat{R}_{\texttt{DIV}}, \{s \mid \texttt{AtStart}(s) \wedge P(s)\}, \{\texttt{AtEnd}, Q\})$ (where $\hat{R}_{\texttt{DIV}}$ as in Q1, and `AtStart`, `AtStart` are as in the slides).

The desired property ϕ, expressing that if `DIV` is run in a state satisfying atomic property P and if it terminates, then in the state in which it terminates atomic property Q holds, is then $\lambda s.\ \texttt{AtEnd}(s) \Rightarrow Q(s)$.

One cannot represent the property that `DIV` terminates in the form $M \models \mathbf{GA}\phi$ because we need to express $\exists s'.\ s' \in \mathsf{Reachable}\ M \wedge \texttt{AtEnd}(s')$. If $\mathsf{Reachable}\ M$ were empty then $M \models \mathbf{GA}\phi$ is (vacuously) true for any ϕ, but the existential property $\exists s'.\ s' \in \mathsf{Reachable}\ M \wedge \texttt{AtEnd}(s')$ is false.

Q5

Disjunctive partitioning can sometimes be used to avoid having to build the BDD of a transition relation when symbolically computing the set of reachable states.

Explain how it might also be used to avoid building the BDD of the transition relation when generating traces to counterexamples. Illustrate you answer using the transition relation R defined by:

$$
\begin{array}{l}
R\ (x,y,z)\ (x',y',z') = \\
\quad (x' = \delta_x(x,y,z) \quad \wedge \quad y' = y \quad \wedge \quad z' = z) \vee \\
\quad (x' = x \quad \wedge \quad y' = \delta_y(x,y,z) \quad \wedge \quad z' = z) \vee \\
\quad (x' = x \quad \wedge \quad y' = y \quad \wedge \quad z' = \delta_z(x,y,z))
\end{array}
$$

Solution

When iterating backwards from a counterexample to generate a path to it, the transition relation R is needed at step i to find an instantiation b_{i-1} for s_{i-1} that makes $s_{i-1} \in \mathcal{S}_{i-1} \wedge R\ s_{i-1}\ b_i$ true.

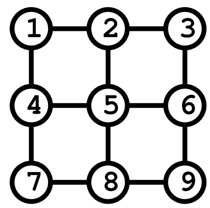
When working symbolically using BDDs one uses SAT on the BDD representing a formula of the form: $h_{i-1}(\vec{v}) \wedge R\ \vec{v}\ \vec{b}_i$, where $h_i(\vec{v})$ represents $s_i \in \mathcal{S}_i$. Taking $\vec{v}$, $\vec{b}_i$ to be (x,y,z), (b_{i1}, b_{i2}, b_{i3}):

$$
\begin{array}{ll}
h_{i-1}(x,y,z) & \wedge\ R\ (x,y,z)\ (b_{i1},b_{i2},b_{i3}) \\
= & \\
h_{i-1}(x,y,z) & \wedge\ (b_{i1} = \delta_x(x,y,z) \quad \wedge \quad b_{i2} = y \quad \wedge \quad b_{i3} = z) \vee \\
& \quad (b_{i1} = x \quad \wedge \quad b_{i2} = \delta_y(x,y,z) \quad \wedge \quad b_{i3} = z) \vee \\
& \quad (b_{i1} = x \quad \wedge \quad b_{i2} = y \quad \wedge \quad b_{i3} = \delta_z(x,y,z)) \\
= & \\
(h_{i-1}(x,y,z) & \wedge\ b_{i1} = \delta_x(x,y,z) \quad \wedge \quad b_{i2} = y \quad \wedge \quad b_{i3} = z) \vee \\
(h_{i-1}(x,y,z) & \wedge\ b_{i1} = x \quad \wedge \quad b_{i2} = \delta_y(x,y,z) \quad \wedge \quad b_{i3} = z) \vee \\
(h_{i-1}(x,y,z) & \wedge\ b_{i1} = x \quad \wedge \quad b_{i2} = y \quad \wedge \quad b_{i3} = \delta_z(x,y,z)) \\
= & \\
(h_{i-1}(x,b_{i2},b_{i3}) & \wedge\ b_{i1} = \delta_x(x,b_{i2},b_{i3}) \quad \wedge \quad b_{i2} = y \quad \wedge \quad b_{i3} = z) \vee \\
(h_{i-1}(b_{i1},y,b_{i3}) & \wedge\ b_{i1} = x \quad \wedge \quad b_{i2} = \delta_y(b_{i1},y,b_{i3}) \quad \wedge \quad b_{i3} = z) \vee \\
(h_{i-1}(b_{i1},b_{i2},z) & \wedge\ b_{i1} = x \quad \wedge \quad b_{i2} = y \quad \wedge \quad b_{i3} = \delta_z(b_{i1},b_{i2},z))
\end{array}
$$

Computing the BDD of the formula above does not require computing the full BDD of R: one can separately compute the BDDs of $\delta_x(x,y,z)$, $\delta_y(x,y,z)$ and $\delta_z(x,y,z)$ and then instantiate them for the occurrences in the formula.

Q6

This questions concerns the nine switches puzzle in the slides:



By defining a state transition function δ_i for each switch ($1 \leq i \leq 9$) express the transition relation `Trans` as the asynchronous interleaving semantics of nine state machines in parallel.

Comment on how this might help with solving the problem by symbolic model checking.

Solution

Define:

$$\delta_1(\texttt{v1,v2,v4}) = (\neg\texttt{v1},\neg\texttt{v2},\neg\texttt{v4})$$

$$\delta_2(\texttt{v1,v2,v3,v5}) = (\neg\texttt{v1},\neg\texttt{v2},\neg\texttt{v3},\neg\texttt{v5})$$

$\vdots$

then

$$\begin{array}{ll}
\multicolumn{2}{l}{\texttt{Trans(v1,v2,v3,v4,v5,v6,v7,v8,v9)(v1',v2',v3',v4',v5',v6',v7',v8',v9')}} \\
\;= (((\texttt{v1',v2',v4'})=\delta_1(\texttt{v1,v2,v4})) & \\
\quad\wedge(\texttt{v3'=v3})\wedge(\texttt{v5'=v5})\wedge(\texttt{v6'=v6})\wedge(\texttt{v7'=v7})\wedge(\texttt{v8'=v8})\wedge(\texttt{v9'=v9})) & \text{(toggle switch 1)} \\
\;\vee\; (((\texttt{v1',v2',v3',v5'})=\delta_2(\texttt{v1,v2,v3,v5})) & \\
\quad\wedge(\texttt{v4'=v4})\wedge(\texttt{v6'=v6})\wedge(\texttt{v7'=v7})\wedge(\texttt{v8'=v8})\wedge(\texttt{v9'=v9})) & \text{(toggle switch 2)}
\end{array}$$

$\vdots$

and hence (with a bit of logical simplification)

$$\begin{array}{l}
\exists\overline{\texttt{v1}}\ \overline{\texttt{v2}}\ \overline{\texttt{v3}}\ \overline{\texttt{v4}}\ \overline{\texttt{v5}}\ \overline{\texttt{v6}}\ \overline{\texttt{v7}}\ \overline{\texttt{v8}}\ \overline{\texttt{v9}}. \\
\;\texttt{f}(\overline{\texttt{v1}},\overline{\texttt{v2}},\overline{\texttt{v3}},\overline{\texttt{v4}},\overline{\texttt{v5}},\overline{\texttt{v6}},\overline{\texttt{v7}},\overline{\texttt{v8}},\overline{\texttt{v9}}) \wedge \texttt{Trans}(\overline{\texttt{v1}},\overline{\texttt{v2}},\overline{\texttt{v3}},\overline{\texttt{v4}},\overline{\texttt{v5}},\overline{\texttt{v6}},\overline{\texttt{v7}},\overline{\texttt{v8}},\overline{\texttt{v9}})(\texttt{v1,v2,v3,v4,v5,v6,v7,v8,v9}) \\
\;= \\
\;(\exists\overline{\texttt{v1}}\ \overline{\texttt{v2}}\ \overline{\texttt{v4}}.\ \texttt{f}(\overline{\texttt{v1}},\overline{\texttt{v2}},\texttt{v3},\overline{\texttt{v4}},\texttt{v5,v6,v7,v8,v9}) \wedge ((\texttt{v1,v2,v4})=\delta_1(\overline{\texttt{v1}},\overline{\texttt{v2}},\overline{\texttt{v4}}))) \\
\;\vee \\
\;(\exists\overline{\texttt{v1}}\ \overline{\texttt{v2}}\ \overline{\texttt{v3}}\ \overline{\texttt{v5}}.\ \texttt{f}(\overline{\texttt{v1}},\overline{\texttt{v2}},\overline{\texttt{v3}},\texttt{v4},\overline{\texttt{v5}},\texttt{v6,v7,v8,v9}) \wedge ((\texttt{v1,v2,v3,v5})=\delta_2(\overline{\texttt{v1}},\overline{\texttt{v2}},\overline{\texttt{v3}},\overline{\texttt{v5}})))
\end{array}$$

$\vdots$

This shows that disjunctive partitioning might work. Actually, one doesn't need to define the δ_i to perform partitioning - one can perform it directly ... so defining the δ_i transition functions doesn't really help.

Q7

Consider the following board which is meant to represent the initial state of the puzzle Peg Solitaire.

```
                ------------------------
                | xxxxx | xxxxx | xxxxx |
                ------------------------
                | xxxxx | xxxxx | xxxxx |
--------------------------------------------------------
| xxxxx | xxxxx | xxxxx | xxxxx | xxxxx | xxxxx | xxxxx |
--------------------------------------------------------
| xxxxx | xxxxx | xxxxx |       | xxxxx | xxxxx | xxxxx |
--------------------------------------------------------
| xxxxx | xxxxx | xxxxx | xxxxx | xxxxx | xxxxx | xxxxx |
--------------------------------------------------------
                | xxxxx | xxxxx | xxxxx |
                ------------------------
                | xxxxx | xxxxx | xxxxx |
                ------------------------
```

All the positions in the board, except the one in the middle, are occupied by pegs, denoted by xxxxx. A move consists of 'jumping' a peg over an adjacent peg in the same row or column into a hole, and removing the peg that was jumped over from the board (thereby reducing the number of pegs on the board by one). The puzzle is to find a sequence of moves, starting from the above configuration, to a configuration consisting of just one peg in the middle, i.e.:

```
                ------------------------
                |       |       |       |
                ------------------------
                |       |       |       |
--------------------------------------------------------
|       |       |       |       |       |       |       |
--------------------------------------------------------
|       |       |       | xxxxx |       |       |       |
--------------------------------------------------------
|       |       |       |       |       |       |       |
--------------------------------------------------------
                |       |       |       |
                ------------------------
                |       |       |       |
                ------------------------
```

Describe how you could formulate Peg Solitaire as the problem of computing the set of reachable states.

Would disjunctive partitioning be useful?

Hint: Your answers to the previous two questions might be useful.

Solution

Peg Solitaire can easily formulated as a state exploration problem by assigning a boolean variable to each board position, for example:

```
                    -------------------------
                    |  v00  |  v02  |  v03  |
                    -------------------------
                    |  v04  |  v05  |  v06  |
    ---------------------------------------------------------
    |  v07  |  v08  |  v09  |  v10  |  v11  |  v12  |  v13  |
    ---------------------------------------------------------
    |  v14  |  v15  |  v16  |  v17  |  v18  |  v19  |  v20  |
    ---------------------------------------------------------
    |  v21  |  v22  |  v23  |  v24  |  v25  |  v26  |  v27  |
    ---------------------------------------------------------
                    |  v28  |  v29  |  v30  |
                    -------------------------
                    |  v31  |  v32  |  v33  |
                    -------------------------
```

The initial state is represented by

```
v01 ∧ v02 ∧ ... ∧ v16 ∧ ¬v17 ∧ v18 ∧ ... ∧ v33
```

and the final(goal) state by

```
¬v01 ∧ ¬v02 ∧ ... ∧ ¬v16 ∧ v17 ∧ ¬v18 ∧ ... ∧ ¬v33
```

The transition relation, say `R`, is then defined to be a disjunctions of terms, with one disjunct per possible move, so that `R((v01,...,v33),(v01',...,v33'))` is true if and only if there exists a move in state `(v01,...,v33)` that results in state `(v01',...,v33')`. Inspection of the board shows that there are 76 moves. The term for a move specifies a change to three variable and that the remaining 30 variables are unchanged. For example, the term

```
v05 ∧ v10 ∧ ¬v17 ∧ ¬v05' ∧ ¬v10' ∧ v17' ∧
(v01' = v01) ∧ (v02' = v02) ∧ (v03' = v03) ∧ (v04' = v04) ∧
(v06' = v06) ∧ (v07' = v07) ∧ (v08' = v08) ∧ (v09' = v09) ∧
(v11' = v11) ∧ (v12' = v12) ∧ (v13' = v13) ∧ (v14' = v14) ∧
(v15' = v15) ∧ (v16' = v16) ∧ (v18' = v18) ∧ (v19' = v19) ∧
(v20' = v20) ∧ (v21' = v21) ∧ (v22' = v22) ∧ (v23' = v23) ∧
(v24' = v24) ∧ (v25' = v25) ∧ (v26' = v26) ∧ (v27' = v27) ∧
(v28' = v28) ∧ (v29' = v29) ∧ (v30' = v30) ∧ (v31' = v31) ∧
(v32' = v32) ∧ (v33' = v33)
```

specifies that the peg at position `05` jump over the peg at position `10` into the centre hole (i.e. the one at position `17`), and all other positions remain unchanged.

Standard BDD methods can be used to compute a sequence of states starting with the initial state, ending with the final state, and such that adjacent elements in the sequance satisfy the transition relation `R`.

Since the transition relation is a disjunction of conjunctions, with many conjuncts specifying that a state variable is unchanged (primed = unprimed), it is likely that disjunctive partitioning would greatly reduce the size of the BDDs needed to symbolically compute the set of reachable states.

Q8

This question concerns the 2-thread program `JM1` described in the slides.

Thread 1	Thread 2
`0:  IF LOCK=0 THEN LOCK:=1;`	`0:  IF LOCK=0 THEN LOCK:=1;`
`1:  X:=1;`	`1:  X:=2;`
`2:  IF LOCK=1 THEN LOCK:=0;`	`2:  IF LOCK=1 THEN LOCK:=0;`
`3:`	`3:`

Draw the computation tree of states $(pc_1, pc_2, lock, x)$ of `JM1` starting at state $(0, 0, 0, 0)$ (i.e. $pc_1 = 0 \wedge pc_2 = 0 \wedge lock = 0 \wedge x = 0$).

Let $M_{\texttt{JM1}} = (S_{\texttt{JM1}}, \{(0, 0, 0, 0)\}, R_{\texttt{JM1}}, AP)$, where $R_{\texttt{JM1}}$ is a total[1] transition relation corresponding to your computation tree and AP contains all atomic properties of the form $\langle x{=}v\rangle$ which mean state component x has value v (e.g. $\langle lock{=}0\rangle$ means $(\lambda(pc_1, pc_2, lock, x).\ lock = 0)$).

Explain the meaning of each of the following LTL properties and say whether it is true.

$M_{\texttt{JM1}} \models \mathbf{F}\langle pc_1{=}3\rangle$
$M_{\texttt{JM1}} \models \mathbf{G}(\langle lock{=}1\rangle \Rightarrow \mathbf{F}\langle lock{=}0\rangle)$
$M_{\texttt{JM1}} \models \mathbf{G}(\langle pc_1{=}2\rangle \Rightarrow \mathbf{X}\langle pc_1{=}3\rangle)$
$M_{\texttt{JM1}} \models \mathbf{F}(\langle pc_1{=}1\rangle \wedge \langle pc_2{=}1\rangle)$
$M_{\texttt{JM1}} \models \mathbf{G}(\langle pc_1{=}3\rangle \Rightarrow \mathbf{G}\langle pc_1{=}3\rangle)$

Explain the meaning of each of the following CTL properties and say whether it is true.

$M_{\texttt{JM1}} \models \mathbf{EF}\langle pc_1{=}3\rangle$
$M_{\texttt{JM1}} \models \mathbf{EFAF}\langle x{=}1\rangle$
$M_{\texttt{JM1}} \models \mathbf{EF}(\langle lock{=}0\rangle \wedge \langle x{=}1\rangle)$
$M_{\texttt{JM1}} \models \mathbf{E}[\langle lock{=}0\rangle \mathbf{U} \langle x{=}2\rangle]$

Explain the meaning of each of the following CTL* properties and say whether it is true.

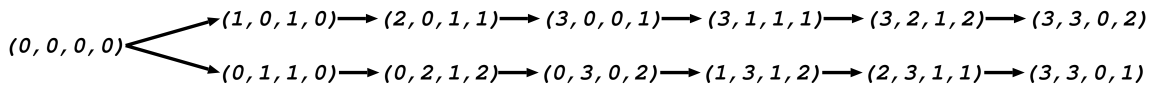
$M_{\texttt{JM1}} \models \mathbf{A}(\mathbf{FG}\langle lock{=}0\rangle \vee \mathbf{F}\langle x{=}2\rangle)$
$M_{\texttt{JM1}} \models \mathbf{E}(\mathbf{X}\langle pc_1{=}1\rangle \wedge \mathbf{F}\langle pc_1{=}3\rangle)$
$M_{\texttt{JM1}} \models \mathbf{A}(\mathbf{X}\langle pc_1{=}1\rangle \Rightarrow \mathbf{F}\langle pc_1{=}3\rangle)$
$M_{\texttt{JM1}} \models \mathbf{A}(\mathbf{G}(\langle pc_1{=}1\rangle \Rightarrow \mathbf{X}(\mathbf{G}\langle x{=}1\rangle)))$

[1] **Hint:** see Q1.

Solution

[Thanks to Jakub Kaplan for spotting errors in my original solution in which the computation tree was wrong.
There may still be errors - please email Mike if you spot one!]

Here is the computation tree.



$M_{\mathtt{JM1}} \models \mathbf{F}\langle pc_1{=}3\rangle$

Thread 1 eventually gets to line 3. True.

$M_{\mathtt{JM1}} \models \mathbf{G}(\langle lock{=}1\rangle \Rightarrow \mathbf{F}\langle lock{=}0\rangle)$

If *lock* is 1 sometime later it will be 0. True.

$M_{\mathtt{JM1}} \models \mathbf{G}(\langle pc_1{=}2\rangle \Rightarrow \mathbf{X}\langle pc_1{=}3\rangle)$

If Thread 1 reaches line 2 then after the next step it will be at line 3. True.

$M_{\mathtt{JM1}} \models \mathbf{F}(\langle pc_1{=}1\rangle \wedge \langle pc_2{=}1\rangle)$

Both threads can simultaneously be at line 1. False.

$M_{\mathtt{JM1}} \models \mathbf{G}(\langle pc_1{=}3\rangle \Rightarrow \mathbf{G}\langle pc_1{=}3\rangle)$

If Thread 1 reaches line 3, it will stay there. True (if relation appropriately totalised).

$M_{\mathtt{JM1}} \models \mathbf{EF}\langle pc_1{=}3\rangle$

Can reach line 3 of Thread 1. True.

$M_{\mathtt{JM1}} \models \mathbf{EFAF}\langle x{=}1\rangle$

There's an execution with x eventually stuck at 1. True if Thread 1 executed second.

$M_{\mathtt{JM1}} \models \mathbf{EF}(\langle lock{=}0\rangle \wedge \langle x{=}1\rangle)$

Can get to a state with $lock = 0$ and $x = 1$. True: happens in execution of Thread 1.

$M_{\mathtt{JM1}} \models \mathbf{E}[\langle lock{=}0\rangle \mathbf{U} \langle x{=}2\rangle]$

There's an execution with $lock = 0$ until $x = 2$. False.

$M_{\mathtt{JM1}} \models \mathbf{A}(\mathbf{FG}\langle lock{=}0\rangle \vee \mathbf{F}\langle x{=}2\rangle)$

On all executions either *lock* eventually stuck at 0 or $x = 2$. True (*lock* stuck at 0).

$M_{\mathtt{JM1}} \models \mathbf{E}(\mathbf{X}\langle pc_1{=}1\rangle \wedge \mathbf{F}\langle pc_1{=}3\rangle)$

On some execution Thread 1 is next at line 1 and eventually at line 3. True.

$M_{\mathtt{JM1}} \models \mathbf{A}(\mathbf{X}\langle pc_1{=}1\rangle \Rightarrow \mathbf{F}\langle pc_1{=}3\rangle)$

On all executions if Thread 1 is next at line 1 then eventually it is at line 3. True.

$M_{\mathtt{JM1}} \models \mathbf{A}(\mathbf{G}(\langle pc_1{=}1\rangle \Rightarrow \mathbf{X}(\mathbf{G}\langle x{=}1\rangle)))$

On all executions if Thread 1 is at line 1 then after one step $x = 1$. False.

Q9

This question uses the 2-thread program JM1 described in the slides (and also used in the preceding question).

In the slide on model checking $\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]$ the set of marked states $\{\!\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!\!\}$ is defined by:

$$\{\!\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!\!\} = \bigcup_{n=0}^{\infty} \{\!\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!\!\}_n$$

For the model $M_{\mathtt{JM1}}$, calculate $\{\!\![\mathbf{E}[\langle lock{=}0\rangle \mathbf{U} \langle x{=}2\rangle]]\!\!\}$ and $\{\!\![\mathbf{E}[\langle pc_1{=}0\rangle \mathbf{U} \langle x{=}2\rangle]]\!\!\}$ and explain how this is used to check:

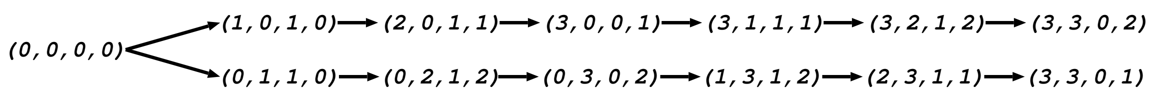
$M_{\mathtt{JM1}} \models \mathbf{E}[\langle lock{=}0\rangle \mathbf{U} \langle x{=}2\rangle]$
$M_{\mathtt{JM1}} \models \mathbf{E}[\langle pc_1{=}0\rangle \mathbf{U} \langle x{=}2\rangle]$

Are either of these true? Explain your answer.

Solution

[Warning: there may be errors - email Mike if you spot one!]

Here is the computation tree.



From this tree

$$\{\!\![\mathbf{E}[\langle lock{=}0\rangle \mathbf{U} \langle x{=}2\rangle]]\!\!\}_0 \;=\; \{\!\![\langle x{=}2\rangle]\!\!\} \;=\; \{(3,2,1,2),(3,3,0,2),(0,2,1,2),(0,3,0,2),(1,3,1,2)\}$$

Also from the tree it can be seen that there are additional states satisfying $\langle lock{=}0\rangle$ with an $R_{\mathtt{JM1}}$-successor in $\{\!\![\mathbf{E}[\langle lock{=}0\rangle \mathbf{U} \langle x{=}2\rangle]]\!\!\}_0$. Thus:

$$\{\!\![\mathbf{E}[\langle lock{=}0\rangle \mathbf{U} \langle x{=}2\rangle]]\!\!\} = \{(3,2,1,2),(3,3,0,2),(0,2,1,2),(0,3,0,2),(1,3,1,2)\}$$

Also from the tree:

$\{\!\![\mathbf{E}[\langle pc_1{=}0\rangle \mathbf{U} \langle x{=}2\rangle]]\!\!\}_0 = \{(3,2,1,2),(3,3,0,2),(0,2,1,2),(0,3,0,2),(1,3,1,2)\}$
$\{\!\![\mathbf{E}[\langle pc_1{=}0\rangle \mathbf{U} \langle x{=}2\rangle]]\!\!\}_1 = \{(3,2,1,2),(3,3,0,2),(0,2,1,2),(0,3,0,2),(1,3,1,2),(0,1,1,0)\}$
$\{\!\![\mathbf{E}[\langle pc_1{=}0\rangle \mathbf{U} \langle x{=}2\rangle]]\!\!\}_2 = \{(3,2,1,2),(3,3,0,2),(0,2,1,2),(0,3,0,2),(1,3,1,2),(0,1,1,0),(0,0,0,0)\}$
$\{\!\![\mathbf{E}[\langle pc_1{=}0\rangle \mathbf{U} \langle x{=}2\rangle]]\!\!\}_n = \{(3,2,1,2),(3,3,0,2),(0,2,1,2),(0,3,0,2),(1,3,1,2),(0,1,1,0),(0,0,0,0)\}(n{>}2)$
$\{\!\![\mathbf{E}[\langle pc_1{=}0\rangle \mathbf{U} \langle x{=}2\rangle]]\!\!\} = \{(3,2,1,2),(3,3,0,2),(0,2,1,2),(0,3,0,2),(1,3,1,2),(0,1,1,0),(0,0,0,0)\}$

If $M_{\mathtt{JM1}} = (S_{\mathtt{JM1}}, \{(0,0,0,0)\}, R_{\mathtt{JM1}}, AP\})$ then $M_{\mathtt{JM1}} \models \phi \Leftrightarrow \{(0,0,0,0)\} \subseteq \{\!\![\phi]\!\!\}$. Thus $M_{\mathtt{JM1}} \models \mathbf{E}[\langle lock{=}0\rangle \mathbf{U} \langle x{=}2\rangle]$ is false and $M_{\mathtt{JM1}} \models \mathbf{E}[\langle pc_1{=}0\rangle \mathbf{U} \langle x{=}2\rangle]$ is true.

Q10

Consider the program `DIV` used in the slides:

```
0:  R:=X;
1:  Q:=0;
2:  WHILE Y≤R DO
3:  (R:=R-Y;
4:   Q:=Q+1)
5:
```

Suppose the program variables `X`, `Y`, `Q`, `R` are restricted to natural numbers less than 256.

Explain how you might represent sets of states and the transition relation as BDDs suitable for use in symbolic model checking. You need not give full details, but should describe how such details would be generated.

Solution

In the slides model $(S_{\texttt{DIV}}, R_{\texttt{DIV}})$ is specified by:

$S_{\texttt{DIV}} = [0..5] \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ (where $[m..n] = \{m, m{+}1, \ldots, n\}$)

$$
\begin{array}{l}
R_{\texttt{DIV}}\ (pc, x, y, r, q)\ (pc', x', y', r', q') = \\
\quad (pc = 0) \Rightarrow ((pc', x', y', r', q') = (1, x, y, x, q)) \quad \wedge \\
\quad (pc = 1) \Rightarrow ((pc', x', y', r', q') = (2, x, y, r, 0)) \quad \wedge \\
\quad (pc = 2) \Rightarrow ((pc', x', y', r', q') = ((\mathit{if}\ y{\leq}r\ \mathit{then}\ 3\ \mathit{else}\ 5), x, y, r, q)) \quad \wedge \\
\quad (pc = 3) \Rightarrow ((pc', x', y', r', q') = (4, x, y, (r{-}y), q)) \quad \wedge \\
\quad (pc = 4) \Rightarrow ((pc', x', y', r', q') = (2, x, y, r, (q{+}1)))
\end{array}
$$

Any number in $[0..5]$ can be represented in binary by a 3-bit word. If program variables are natural numbers less than 256, then they can be represented in binary by 8-bit words, so the state space could instead be:

$$S_{\texttt{DIV}} \ = \ \mathbb{B}^3 \times \mathbb{B}^8 \times \mathbb{B}^8 \times \mathbb{B}^8 \times \mathbb{B}^8 \ \cong \ \mathbb{B}^{35}$$

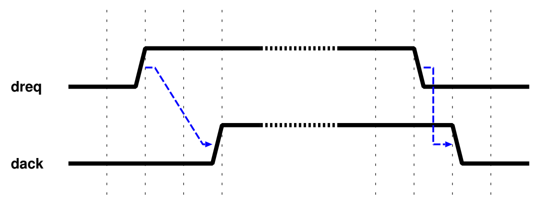
Thus a state can be represented symbolically by a Boolean formula with 35 variables. To define the transition relation one needs to define binary versions of addition (+) and subtraction (−). This can be done naively by using adder and subtractor hardware circuits as the basis of Boolean functions:

$\mathsf{BinaryAdd}((m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7), (n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7))$
$\mathsf{BinarySub}((m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7), (n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7))$

that return appropriate 8-bit words as results. One also needs to define a Boolean function to compute 8-bit less-then-or-equal ($\leq$).

Q11

The timing diagram below is mentioned in the slides.



The following two handshake properties were given:

- following a rising edge on dreq, the value of dreq remains 1 (i.e. *true*) until it is acknowledged by a rising edge on dack

- following a falling edge on dreq, the value on dreq remains 0 (i.e. *false*) until the value of dack is 0

Formalise these two properties as formulae in a suitable temporal logic. You should state what logic you are using and briefly describe why you chose it.

Solution

I will use PSL because it is more readable, though LTL would do. If we interpret the informal text as requiring that a dack must always occur, the properties are:

```
always (dreq  -> (dreq  until! dack))
always (!dreq -> (!dreq until! !dack))
```

This is a strong until ([dreg **U** dack] in LTL). If we don't require an acknowlegement to actually happen, then a weak until ([dreg **W** dack] in LTL) would be used:

```
always (dreq  -> (dreq  until dack))
always (!dreq -> (!dreq until !dack))
```

The timing diagram, but not the informal textual descriptions, could be understood as implying that dack should be required to be low during the rising edge and high during the falling edge. This could be expressed, for the strong interpretation, by:

```
always {!dreq;dreq} |-> {!dack:dreq[*]:dack}
always {dreq;!dreq} |-> {dack:!dreq[*]:!dack}
```

[Thanks to Cindy Eisner for comments on and corrections to an earlier solution.]

Q12

The DIV example is discussed in the slides:

```
0:   R:=X;
1:   Q:=0;
2:   WHILE Y≤R DO
3:   (R:=R-Y;
4:    Q:=Q+1)
5:
```

AtStart (pc, x, y, r, q)	=	$(pc = 0)$
AtEnd (pc, x, y, r, q)	=	$(pc = 5)$
InLoop (pc, x, y, r, q)	=	$(pc \in \{3, 4\})$
YleqR (pc, x, y, r, q)	=	$(y \leq r)$
Invariant (pc, x, y, r, q)	=	$(x = r + (y \times q))$

The following three properties were given:

- on every execution if AtEnd is true then Invariant is true and YleqR is not true
- on every execution there is a state where AtEnd it true
- on any execution if there exists a state where YleqR is true then there is also a state where InLoop is true

Formalise these three properties as formulae in a suitable temporal logic. You should state what logic you are using and briefly describe why you chose it.

Solution

Since the properties specify *all* program executions, and executions can be represented as paths starting from an initial state $(0, x, y, r, q)$, it is natural to represent the properties by CTL formulae evaluated at initial states $(0, x, y, r, q)$, where the ψs for the three properties are:

$\mathbf{AG}(\texttt{AtEnd} \Rightarrow (\texttt{Invariant} \wedge \neg\texttt{YleqR}))$

$\mathbf{AF}\ \texttt{AtEnd}$

$(\mathbf{AF}\ \texttt{YleqR}) \Rightarrow (\mathbf{AF}\ \texttt{InLoop})$

However, the corresponding LTL would also be fine:

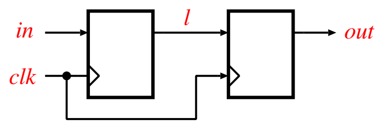
$\mathbf{G}(\texttt{AtEnd} \Rightarrow (\texttt{Invariant} \wedge \neg\texttt{YleqR}))$

$\mathbf{F}\ \texttt{AtEnd}$

$(\mathbf{F}\ \texttt{YleqR}) \Rightarrow (\mathbf{F}\ \texttt{InLoop})$

Q13

The following circuit consisting of two Dtype in series is mentioned in the slides.



The behaviour of this was described informally the giving the trace:

```
in   aaaaaaaaaaabbbbbbccccccddddddddd......
clk 00000111110000011111000001111100......
l    eeeeeaaaaaaaaaabbbbbbbbbbddddddd......
out fffffeeeeeeeeeeaaaaaaaaaabbbbbbb......
```

Call this example `D2`. Devise a model to represent `D2` suitable for use in model checking.

Hint: include in the state components for both the current value of *clk* and for its value at the previous time instant, say *prevclk* (e.g. take the state to be $(prevclk, clk, in, l, out)$).

Solution

[Thanks to Jakub Kaplan for spotting errors in my original solution.]

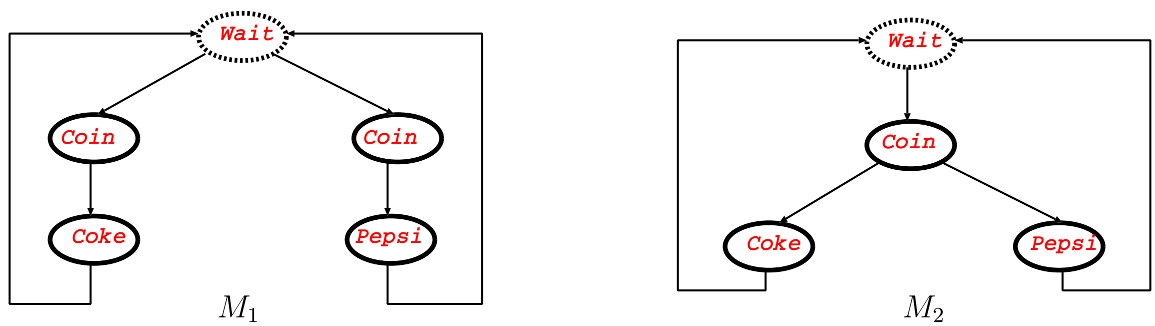
As in the hint, take the set of states to be 5-tuples $(prevclk, clk, in, l, out) \in \mathbb{B}^5$. Define the transition relation $R_{\texttt{D2}}$ by:

$$
\begin{array}{l}
R_{\texttt{D2}}\ (prevclk, clk, in, l, out)\ (prevclk', clk', in', l', out') = \\
(prevclk' = clk) \qquad\qquad\qquad\qquad\quad \wedge \\
(l' = \texttt{if}\ (!prevclk \wedge clk)\ \texttt{then}\ in\ \texttt{else}\ l)\ \wedge \\
(out' = \texttt{if}\ (!prevclk \wedge clk)\ \texttt{then}\ l\ \texttt{else}\ out)
\end{array}
$$

The conjunction $!prevclk \wedge clk$ in the conditionals giving the values of l' and out' is true just after a rising edge of *clk*. Only at such rising edges does the input to the Dtype get latched. At other times - i.e. when there is no rising edge of *clk* - the stored value, which drives the Dtype outputs (i.e. l and *out*), remains unchanged.

Q14

Let models M_1 and M_2 correspond to the state transition diagrams below. Assume $AP = \{Wait, Coin, Coke, Pepsi\}$. Initial states are indicated by a dotted line, and state names are also used to name atomic predicates that only hold at the state with the same name.



Are M_1 and M_2 bisimilar, i.e. $M_1 \equiv M_2$? Justify your answer.

Hint: consider $\mathbf{AG}(Coin \Rightarrow \mathbf{EX}Coke)$.

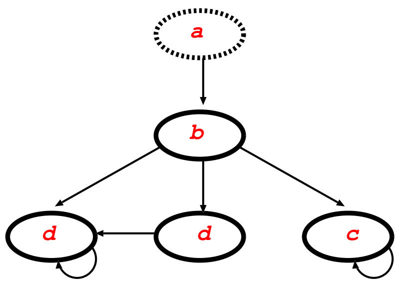
Solution

The CTL formula $\mathbf{AG}(Coin \Rightarrow \mathbf{EX}Coke)$ is false for M_1. However, this formula is true for M_2, hence M_1 and M_2 cannot be bisimilar, since bisimilar models satisfy the same CTL formulae.

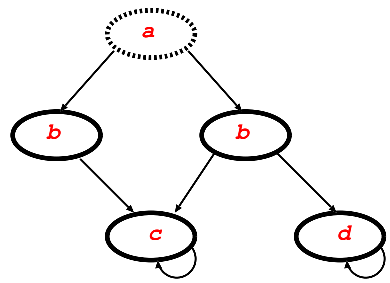
Acknowledgement. The example in this exercise is from online slides for a course by Orna Grumberg.

Q15

Define models M_1 and M_2 that correspond to the state transition diagrams below. The initial states are indicated by a dotted line, and the atomic predicates are shown inside the states where they hold.



M_1

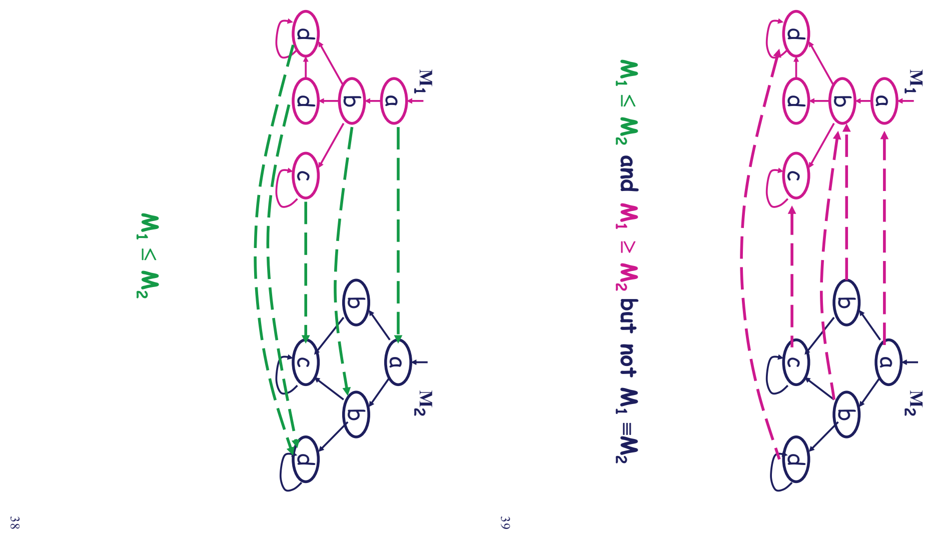


M_2

Does $M_1 \preceq M_2$ or $M_2 \preceq M_1$? Justify your answer. Does it shed light on whether $M_1 \preceq M_2$ and $M_2 \preceq M_1$ entails $M_1 \equiv M_2$?

Solution

Both $M_1 \preceq M_2$ and $M_2 \preceq M_1$, as shown in the following diagrams.



The CTL formula $\mathbf{AG}(b \Rightarrow \mathbf{EX}d)$ is true for M_1 but false for M_2 (consider the right-most state satisfying b), so M_1 and M_2 can't be bisimilar. Thus $M_1 \preceq M_2$ and $M_2 \preceq M_1$ does not entail $M_1 \equiv M_2$?

Acknowledgement. The example in this exercise (including the diagrams in the solution) is from online slides for a course by Orna Grumberg.