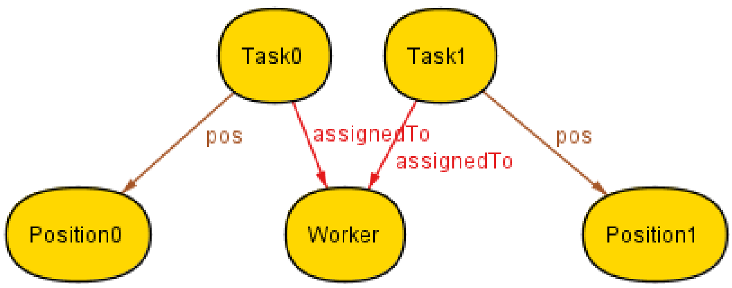
Question 1.a)
Answer: The solution below is based on the standard 'corner case' of an implication: make the implication $(\neg P(f(x))) \rightarrow \forall y.Q(y)$ in φ_2 true by having the lefthand side $\neg P(f(x))$ always be false, i.e., having $P(f(x))$ be true for all x. Assuming that $\forall y.Q(y)$ holds, the resulting model will make φ_2 true, but φ_1 false. As a bonus, it turns out that a one-point domain is enough.
A definition of $\mathcal{M}$ that achieves this is the following: Let $\mathcal{M} = (\mathcal{D}, \mathcal{F}, \mathcal{P})$ where $\mathcal{D} = \{*\}$, $\mathcal{F} = \{f^{\mathcal{M}}\}$, and $\mathcal{P} = \{P^M\}$ such that $f^M(*) = *$, $P^M(*) = \mathit{true}$, and $Q^M(*) = \mathit{true}$.
Then, $\mathcal{M} \not\models \varphi_1$ and $\mathcal{M} \models \varphi_2$.

Question 1.b)
Answer: φ_1 does not hold in $\mathcal{M}$, and φ_2 holds in $\mathcal{M}$.

Question 2.a)
Possible answer:
Instance shown graphically:



Graphical instance represented textually (was not required; included here for completeness):

Name in specification		Interpretation of name in instance
`Task`	=	{ Task0, Task1 }
`Worker`	=	{ Worker }
`Position`	=	{ Position0, Position1}
`pos`	=	{ (Task0,Position0), (Task1,Position1) }
`assignedTo`	=	{ (Task0,Worker), (Task1,Worker) }

Question 2.b)
Answer: 2

Question 2.c)
Possible answers:
`assignedTo.w` or
`w.~assignedTo` or
`{t:Task | w in t.assignedTo}` or
`{t:Task | t.assignedTo = w}`

Question 2.d)
Possible answers:
`{w:Worker | min[getTasks[w].pos] = first}` or
`{w:Worker | some t:getTasks[w] | t.pos = first}` or
`{w:Worker | some t:getTasks[w] | first in t.pos}` or
`{w:Worker | some t:getTasks[w] | all p:Pos | lte[t.pos,p]}`
`{w:Worker | some t:Task | t in getTasks[w] && ...}`

Question 2.e)
Possible answers:
`lt[t1.pos,t2.pos] && lt[t2.pos,t3.pos]`

Question 2.f)
Possible answers:
```
all w1,w2 : Worker  |  #getTasks[w1] = #getTasks[w2]  ||
                               plus[#getTasks[w1],1] = #getTasks[w2]  ||
```

```
                                #getTasks[w1] = plus[#getTasks[w2],1]
```
or
```
all w1,w2 : Worker |  #getTasks[w1] = #getTasks[w2]  ||
                      #getTasks[w1]+1 = #getTasks[w2]  ||
                      #getTasks[w1] = #getTasks[w2]+1
```
or
```
all disj w1,w2 : Worker  |  #getTasks[w1] = #getTasks[w2]  ||
                            plus[#getTasks[w1],1] = #getTasks[w2]  ||
                            #getTasks[w1] = plus[#getTasks[w2],1]
```

Question 2.g)

Possible answers:
```
all w : Worker | all t1,t2 : getTasks[w] |
                      all t3 : Task | between[t1,t2,t3] => t3 in getTasks[w]
```
or
```
all w : Worker | all disj t1,t2 : getTasks[w] |
                      all t3 : Task | between[t1,t2,t3] => t3 in getTasks[w]
```

Question 2.h)

Possible answers:

`all t : Task | (no t' : Task | t->t' in dependsOn) => t in getTasks[getFirstWorker[]]`, or

`all t : Task | (no t' : Task | t' in t.dependsOn) => t in getTasks[getFirstWorker[]]`, or

`all t : Task | no t' : Task | t' in t.dependsOn => t in getTasks[getFirstWorker[]]`, or

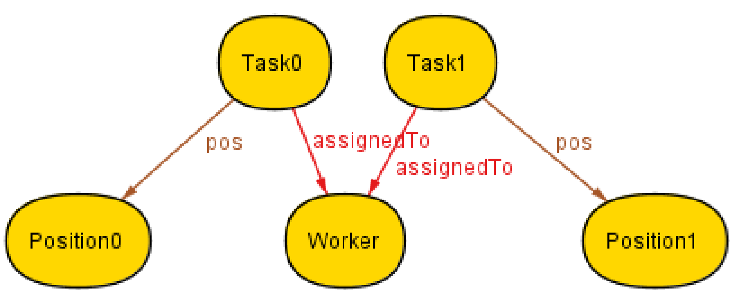
`all t : Task | all t' : Task | !(t' in t.dependsOn) => t in getTasks[getFirstWorker[]]`

Question 2.i)

Answer: Yes, the formula is implied. Fact F2 links the relation `dependsOn` on tasks with the "less than" relation on positions. As a result, `dependsOn` cannot contain any cycles. If it did, then, due to Fact F2, the ordering on positions would be cyclic too, which is impossible.

Question 2.j)

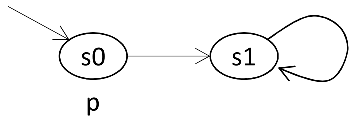
Note that there is a bug in the question (`w.assignedTo` should be replaced by `assignedTo.w`). The answer below assumes this corrected version. Correct student answers based on the buggy version received full marks.
Answer: No, the formula is not implied. The formula says that tasks assigned to a worker must be unique, i.e., that different tasks cannot be assigned to the same worker, i.e., that every worker has at most one task assigned to it. However, there is nothing in the specification `TA` that enforces this. One possible counter example is the following:



Graphical instance represented textually:

Name in specification		Interpretation of name in instance
`Task`	=	{ Task0, Task1 }
`Worker`	=	{ Worker }
`Position`	=	{ Position0, Position1}
`pos`	=	{ (Task0,Position0), (Task1,Position1) }
`assignedTo`	=	{ (Task0,Worker), (Task1,Worker) }

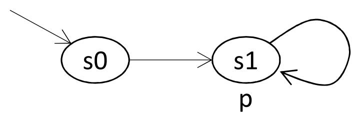
Question 3.a) Answer: Let M_1 be the Kripke structure below. Then, $M_1 \models \varphi_1$ and $M_1 \not\models \varphi'_1$.



Textual representation of same Kripke structure:

$$\begin{array}{rcl} S & = & \{\text{s0, s1}\} \\ S_0 & = & \{\text{s0}\} \\ R & = & \{(\text{s0,s1}),\ (\text{s1,s1})\} \\ L & & \text{such that} \\ & & L(\text{s0}) = \{\ \} \\ & & L(\text{s1}) = \{\ \text{p}\ \} \end{array}$$

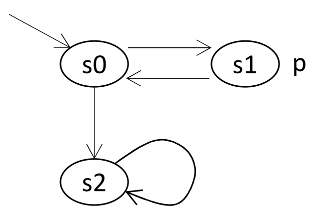
Question 3.b)
Answer: Let M_2 be the Kripke structure below. Then, $M_2 \not\models \varphi_2$ and $M_2 \models \varphi'_2$.



Textual representation of same Kripke structure:

$$\begin{array}{rcl} S & = & \{\text{s0, s1}\} \\ S_0 & = & \{\text{s0}\} \\ R & = & \{(\text{s0,s1}),\ (\text{s1,s1})\} \\ L & & \text{such that} \\ & & L(\text{s0}) = \{\ \} \\ & & L(\text{s1}) = \{\text{p}\} \end{array}$$

Question 3.c)
Answer: Let M_3 be the Kripke structure below. Then, $M_3 \models \varphi_3$ and $M_3 \not\models \varphi'_3$.



Textual representation of same Kripke structure:

$$\begin{array}{rcl} S & = & \{\text{s0, s1, s2}\} \\ S_0 & = & \{\text{s0}\} \\ R & = & \{(\text{s0,s1}),\ (\text{s0,s2}),\ (\text{s1,s0}),\ (\text{s2,s2})\ \} \\ L & & \text{such that} \\ & & L(\text{s0}) = \{\ \} \\ & & L(\text{s1}) = \{\ \text{p}\ \} \\ & & L(\text{s2}) = \{\ \} \end{array}$$

Question 4.a) AX AX AX AX $(q \vee r)$
Answer: "Yes"

Question 4.b) EF EG $\neg p$
Answer: "Yes"

Question 4.c) AG $\big(p \vee (\mathbf{EX}\ p) \vee (\mathbf{EX}\ \mathbf{EX}\ p)\big)$
Answer: "No", $s_0 s_1 s_2$ or $s_0 s_1 s_2 s_1 s_2$ or ...

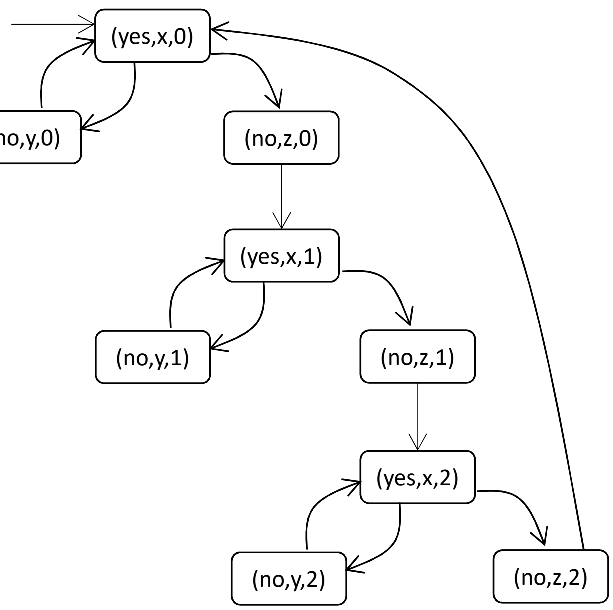
Question 4.d) AG AF $(p \vee q)$

Answer: “No”, $s_0 s_1 (s_2)$ or $s_0 s_1 s_2 s_1 (s_2)$ or ...

Question 4.e) $\mathbf{A}\big[(q \vee \mathbf{EX}\ q)\ \mathbf{U}\ \mathbf{EG}\ r\big]$
Answer: “No”, $(s_0 s_1 s_3)$ (only counter example)

Question 4.f) $\mathbf{AG}\ \big((\mathbf{EX}\ p)\ \rightarrow\ \mathbf{EX}\ \mathbf{AX}\ q\big)$
Answer: “Yes”

Question 5.a)
The line `s=no & c=z & cnt<1 :  cnt+1;` in the SMV code contained a bug and should have read `s=no & c=z & cnt<2 :  cnt+1;`. The Kripke structure shown below belongs to the corrected version. Student answers showing the Kripke structure of the buggy version received full marks. Students who noticed that there was something wrong in the code and drew a correct structure based on some clearly noted correction to the code also received full marks.
Answer:



Textual representation of same Kripke structure:

$$
\begin{array}{rcl}
S & = & \{(yes,x,0),(no,y,0),(no,z,0),(yes,x,1),(no,y,1),(no,z,1),(yes,x,2),(no,y,2),(no,z,2)\} \\
S_0 & = & \{(yes,x,0)\} \\
R & = & \{((yes,x,0),(no,y,0)),((no,y,0),(yes,x,0)),((yes,x,0),(no,z,0)), \\
 & & ((no,z,0),(yes,x,1)), \\
 & & ((yes,x,1),(no,y,1)),((no,y,1),(yes,x,1)),((yes,x,1),(no,z,1)), \\
 & & ((no,z,1),(yes,x,2)), \\
 & & ((yes,x,2),(no,y,2)),((no,y,2),(yes,x,2)),((yes,x,2),(no,z,2)), \\
 & & ((no,z,2),(yes,x,0))\} \\
L & & \text{did not have to be shown (need formula to know what atomic propositions are)}
\end{array}
$$

Note that our definition of the Kripke structure only uses the *reachable* states. The full set of states of the program is

$$S \;=\; \big\{(v_{sig}, v_{choice}, v_{cnt}) \;\mid\; v_{sig} \in \{yes, no\} \wedge v_{choice} \in \{x, y, z\} \wedge v_{cnt} \in \{0, 1, 2\}\big\}$$

and thus contains 18 elements.
Question 5.b)
Answer: The processes execute synchronously.

Question 5.c)
Answer: The program is not deterministic (i.e., it is non-deterministic).
Question 5.d), Part 1
Answer:

A1. $\mathbf{EX}\ \mathbf{EX}\ \mathbf{EX}\ \big(p1.sig = yes \wedge p2.choice = x \wedge p3.cnt = 1\big)$ or
A2. $\neg\mathbf{AX}\ \mathbf{AX}\ \mathbf{AX}\ \neg\big(p1.sig = yes \wedge p2.choice = x \wedge p3.cnt = 1\big)$ or
A3. $\mathbf{AX}\ \mathbf{AX}\ \mathbf{AX}\ \neg\big(p1.sig = yes \wedge p2.choice = x \wedge p3.cnt = 1\big)$

Question 5.d), Part 2
Answer: In case of A1 and A2, we can conclude that "state $(yes, x, 1)$ is reachable in exactly 3 steps". In case of A3, we can conclude that "state $(yes, x, 1)$ is not reachable in exactly 3 steps".
Question 5.e)

φ_1: **Answer:** $\neg\mathbf{EF}\ \big(p1.sig = yes \wedge \mathbf{EX}\ p1.sig = yes\big)$

φ_2: Answer: $\mathbf{AG}\ \mathbf{EF}\ p3.cnt = 0$

φ_3: Answer: $\mathbf{E}\big[p2.choice \neq z\ \mathbf{U}\ \mathbf{EG}\ p3.cnt = 2\big]$

φ_4: Answer: $\mathbf{AX}\ \mathbf{AX}\ \mathbf{AX}\ p3.cnt < 2$

φ_5: Answer: $\mathbf{AG}\ \big(p1.sig = yes\ \rightarrow\ \mathbf{AX}\ p1.sig = no\big)$