Important note: *SVT is a course that provides a formal base for the used concepts, algorithms and tools. This allows you to be* concise, short *and* exact *in your answers. Answers that are unnecessarily long, verbose, or not to-the-point can be subject to point-reduction. If the formulation of a question contains "Argue your answer" and you provide an answer without an argument, then* no *points will be awarded.*

For this exam you may use clean *course materials, notably the workbook and the book "Model Checking. E.M. Clarke, O. Grumberg and D. Peled.". The paper by Tretmans is attached to the exam.*

You can get up to 100 points. The final score of the exam is the number of points divided by 10. You are allowed to use clean *versions of the workbook and the book "Model Checking" by Clarke et al. A version of the paper of Tretmans has been added as appendix to this exam.*

Question 1 **(15 points)**

In the textbook, Clarke et al. provide a translation of programs to Kripke structures (see Section 2.2.2). We are going to add a new statement `either/or` to these programs. Consider the following program code. In the initial state, variables have the following values: $x = 1$, $y = 2$, $z = 0$.

```
while (x > 0 or y > 0)
do
    either do
        x := x + 1;
        y := y - 1
    end
    or do
        x := x - 1;
        y := y + 1
    end
end while;
z := x + y
```

In this program, the `while`-loop executes nondeterministically one of the blocks, until the given condition is true. Thus: each time the loop is entered, it may differ which block is executed.

(a) Extend the translation of Clarke et al. so that it is able to translate `either/or` statements.

Solution: The translation consists of two parts. First, we need to extend the labelling (see Clarke et al. page 21). The program fragment

$$P = \texttt{either do } P_1 \texttt{ end or do } P_2 \texttt{ end}$$

is labelled as

$$P^{\mathcal{L}} = \texttt{either do } l_1\texttt{: } P_1^{\mathcal{L}} \texttt{ end or do } l_2\texttt{: } P_2^{\mathcal{L}} \texttt{ end}$$

Next, we give the translation

$$\mathcal{C}(l, \texttt{either do } l_1\texttt{: } P_1 \texttt{ end or do } l_2\texttt{: } P_2 \texttt{ end}, l')$$

this is defined as the disjunction of the following four formulas:

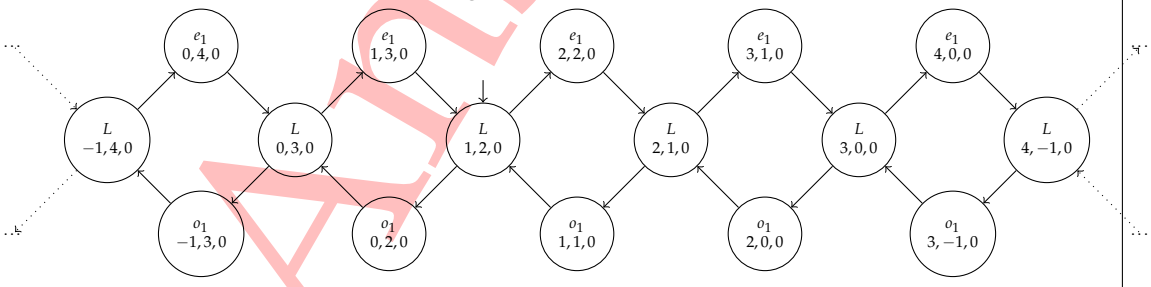
- $pc = l \wedge pc' = l_1 \wedge same(V)$
- $pc = l \wedge pc' = l_2 \wedge same(V)$
- $\mathcal{C}(\ell_1, P_1, l')$
- $\mathcal{C}(\ell_2, P_2, l')$

(b) Provide the Kripke structure that is the result of applying your translation to the given program code.

Solution: We first label the program code.

```
m:   while (x > 0 or y > 0)
     do
w:      either do
e0:          x := x + 1;
e1:          y := y - 1
         end
         or do
o0:          x := x - 1;
o1:          y := y + 1
         end
l:   end while;
m':  z := x + y
```

Next, we build a Kripke structure. Note that the states with labels `m`, `w`, `e0` en `e1` always have the same value for x, y, and z when they occur in a sequence. We therefore take a shortcut and combine these states, and use label L for them. The Kripke structure then becomes the following



When we look at this Kripke structure, there are a few things we should observe.

- In fact, it is not a Kripke structure, since the number of states is infinite, whereas a Kripke structure only has a finite number of states.
- The computation will never terminate, since $x + y = 3$ holds whenever we are in location m, and the guard of the loop will therefore always be satisfied.

This is problematic, and is in fact caused by a typo in the exam question. The guard should have been `x > 0 and y > 0`. This question, and the following *c* and *d* are therefore not taken into account in determining the grade obtained for the exam.

(c) Provide an invariant, i.e., a property of the form $\mathbf{AG}(\phi)$, that does not include an inequality (i.e., does not include either $<$, $\leq$, etc.) and does not include any program labels.

Solution: There are several possibilities for the invariant. Given the Kripke structure above, one invariant is $AG(z = 0)$; alternatively, if the assignment to z would be reachable, the invariant could be $AG(z = 0 \vee z = 3)$.

(d) Does the property **AGAF**$(z > 0)$ hold? Argue your answer.

Solution: Given the Kripke structure in part b of the question, it can be seen that the assignment to z is never reached. Therefore, the property does not hold.

Question 2 **(10 points)**

Consider the following UPPAAL automaton.

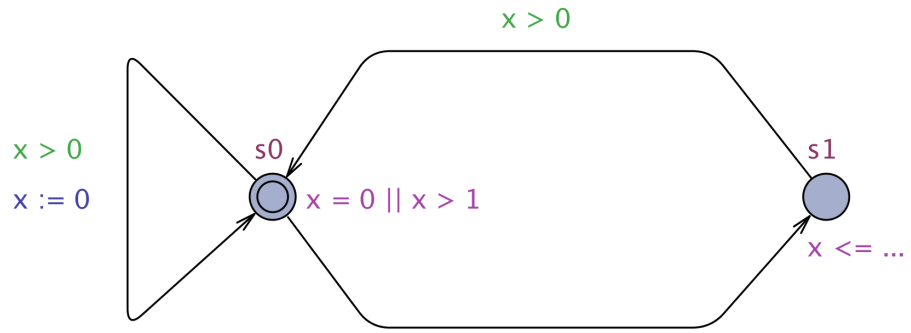


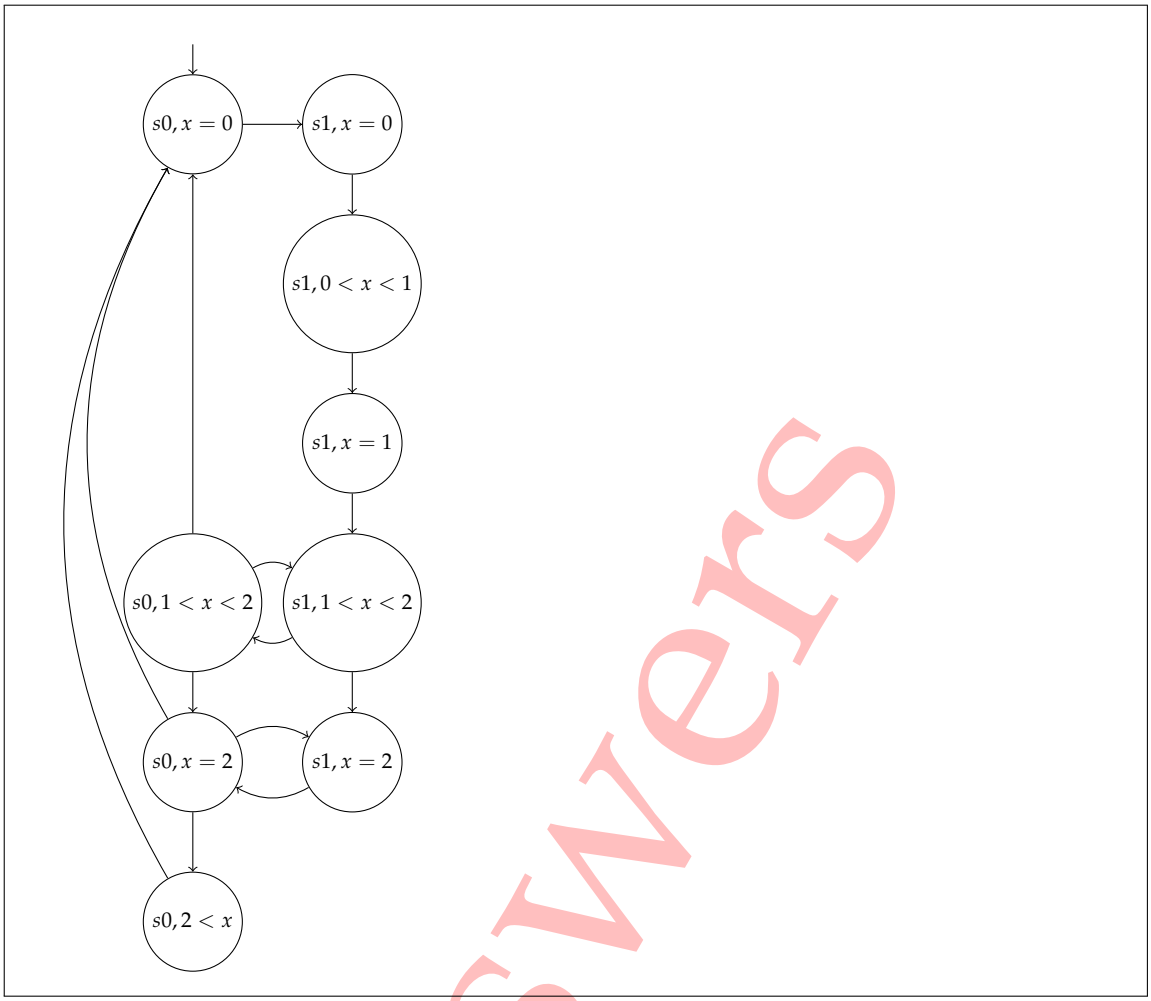
FIGURE 1 UPPAAL automaton. Note that we use := for an assignment in an update, and the = sign for equality in guards and invariants.

(a) What is the lowest natural number that can be filled in at the dots (...), such that the automaton does not have a deadlock? Argue your answer.

Solution: The lowest natural number that can be filled in is 2. If the invariant on $s1$ would be $x \leq 0$, there is a deadlock in $(s1, 0)$ since the guard on the transition to $s1$ is not satisfied, and time cannot elapse. If the invariant on $s1$ would be $x \leq 1$, the transition to $s1$ cannot be taken from $(s1, 1)$ since the invariant on the target location is not satisfied ($x = 0 \vee x > 1$) does not hold. When choosing $x \leq 2$, the guard is satisfied whenever $x > 0$, and the invariant on the target location is satisfied whenever $x > 1$, so in $s1$ the system can always delay to $1 < x \leq 2$ and then take the transition to $s0$.

(b) Draw the corresponding region automaton (assume the dots are replaced by your value).

Solution: For the region automata construction, we need to take into account the largest constant in the timed automaton. Given the answer to part a, the constant is 2. We therefore obtain the following region automaton.



(c) Does the UPPAAL automaton (assume the dots are replaced by your value) exhibit Zeno behavior? Argue your answer. Use the region automaton in your argument.

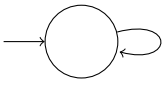
Solution: Yes, the automaton exhibits Zeno behavior. An infinite number of discrete transitions can be taken in a finite amount of time in the cycles between $(s0, 1 < x < 2)$ and $(s1, 1 < x < 2)$, as well as between $(s0, x = 2)$ and $(s1, x = 2)$.

Question 3 **(10 points)**

(a) Provide a Kripke structure that demonstrates the difference between the following CTL formulae. Thus, *exactly* one of these CTL formulae must be true for the Kripke structure.
- $\mathbf{A}(p\ \mathbf{U}\ q)$
- $p \rightsquigarrow q$

Note that $\rightsquigarrow$ is UPPAAL's liveness operator.

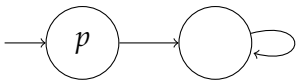
Solution: The following Kripke structure satisfies $p \rightsquigarrow q$, but not $\mathbf{A}(p\ \mathbf{U}\ q)$.



Remark: in the print distributed at the exam, the symbol $\rightsquigarrow$ was missing. Therefore, this part of the question has not been taken into account in the grading of the exam.

(b) Provide a Kripke structure that demonstrates the difference between the following CTL formulae. Thus, *exactly* one of these CTL formulae must be true for the Kripke structure.
- **AGEF**(p)
- **AGEFEX**(p)

Solution: The following Kripke structure satisfies **AGEF**(p), but not **AGEFEX**(p).



Question 4 **(10 points)**

Figure 2 shows a Kripke structure. Execute the algorithm for explicit state model checking step-by-step for the CTL formula:

$$s_0 \models \mathbf{EG}(\mathbf{AF}(p) \vee q)$$

This entails showing each individual step performed, and providing the exact output of the algorithm.

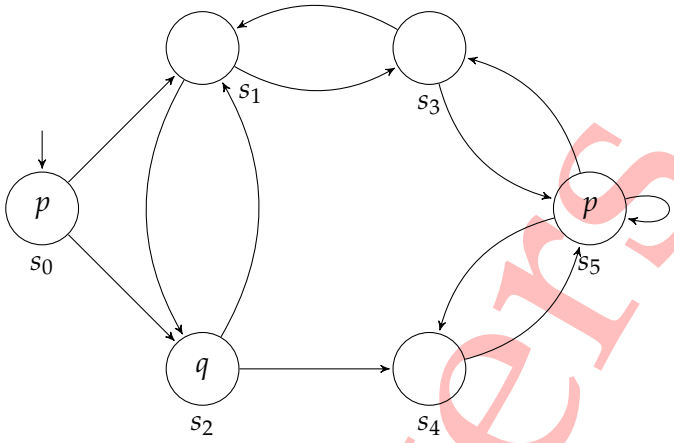


FIGURE 2 Kripke structure

Solution: The explicit model checking algorithm is defined for a subset of CTL. Therefore, first we transform the formula to the appropriate subset, i.e., we get $\mathbf{EG}(\mathbf{AF}(p) \vee q) = \mathbf{EG}(\neg\mathbf{EG}(\neg p) \vee q)$.

Next, we compute the sets S satisfying each of the subformulas, starting with the smallest subformulas.

1. $S(p) = \{s_5\}$
2. $S(q) = \{s_2\}$
3. $S(\neg p) = S \setminus S(p) = \{s_0, s_1, s_2, s_3, s_4\}$
4. Next we compute $S(\mathbf{EG}(\neg p))$. For this, we first determine the non-trivial SCCs of the Kripke structure, restricted to $\{s_0, s_1, s_2, s_3, s_4\}$. This is $\{\{s_0, s_1, s_2\}\}$. All states in this set satisfy $\mathbf{EG}(\neg p)$. Next we apply the backward reachability algorithm to compute the rest of the states, obtaining $S(\mathbf{EG}(\neg p)) = \{s_0, s_1, s_2, s_3\}$.
5. $S(\neg\mathbf{EG}(\neg p)) = S \setminus S(\mathbf{EG}(\neg p)) = S \setminus \{s_0, s_1, s_2, s_3\} = \{s_4, s_5\}$.
6. $S(\neg\mathbf{EG}(\neg p) \vee q) = S(\neg\mathbf{EG}(\neg p)) \cup S(q) = \{s_4, s_5\} \cup \{s_2\} = \{s_2, s_4, s_5\}$.
7. Finally we compute $S(\mathbf{EG}(\neg\mathbf{EG}(\neg p) \vee q))$. We first compute the non-trivial SCCs in the Kripke structure restricted to $\{s_2, s_4, s_5\}$. This is $\{\{s_4, s_5\}\}$. Next, we again perform the backward reachability computation to obtain $\{s_2, s_4, s_5\}$, hence $S(\mathbf{EG}(\neg\mathbf{EG}(\neg p) \vee q)) = \{s_2, s_4, s_5\}$.

Question 5 **(10 points)**

Consider the oven example on page 39 of the textbook. Provide CTL formulations of the following specifications. Use atomic propositions to characterize states, instead of using the state-numbering in the book.

(a) It is possible that the oven will be permanently in an error state.

Solution: $\mathbf{EFEG}(\mathit{Error})$

(b) It is possible that the oven will be permanently stuck in an error state.

Solution: $\mathbf{EFAG}(\mathit{Error})$

(c) The door will eventually be closed.

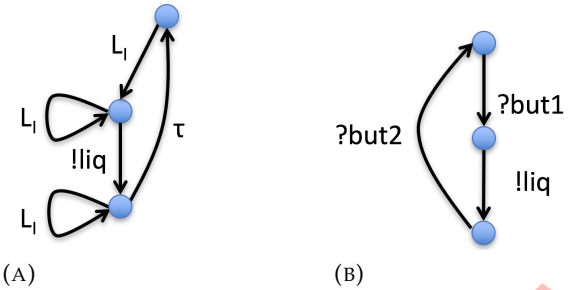
Solution: $\mathbf{AF}(\mathit{Close})$

(d) In any infinite run, the door will infinitely often be closed.

Solution: $\neg \mathit{Close} \rightsquigarrow \mathit{Close}$
An alternative solution that was accepted is $\mathbf{AGAF}(\mathit{Close})$

Question 6 **(10 points)**

Figure 3 shows two IO automata: an implementation i (left) and a specification s (right). The initial states are the upper ones.



(A) (B)

FIGURE 3 Input alphabet $L_I = \{?but1, ?but2\}$. Output alphabet $L_U = \{!liq\}$

(a) Does the implementation conform to the specification, according to the **ioco** definition? Argue your answer.

Solution: No, the implementation does not conform to the specification. For i **ioco** s to hold, we need to have $\forall \sigma \in Straces(s) : out(i \textbf{ after } \sigma) \subseteq out(s \textbf{ after } \sigma)$. Now, consider for instance the suspension trace $\sigma = ?but1 \cdot !liq \cdot ?but2$.

Let us, for ease of reference, number the states from top to bottom, so we get states i_0, i_1, i_2 and s_0, s_1, s_2.

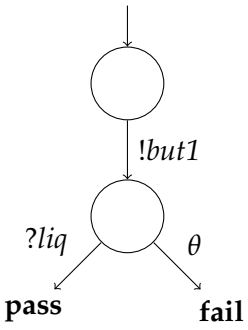
We get the following after sets:
- i **after** $\sigma = \{i_0, i_1, i_2\}$ (note that in the definitions of after, (suspension) traces are used, and the τ is ignored, see Definition 5 on p.9 of Tretmans' paper).
- s **after** $\sigma = \{s_0\}$.

Next we compute the output sets:
- $out(i \textbf{ after } \sigma) = \{!liq, \delta\}$
- $out(s \textbf{ after } \sigma) = \{\delta\}$

Hence, $out(i \textbf{ after } \sigma) \not\subseteq out(s \textbf{ after } \sigma)$, and thus not i **ioco** s.

(b) When the specification is used for model-based testing, test-cases are generated. Give an example of a test case, that includes at least the following:
pass, *fail*, θ, !, and ?

Solution: The simplest test case that can be generated only provides one input, and then checks the outputs. We obtain the following test case:



Note that the course material is not consistent with respect to inputs and outputs, so a correct test case in which input and output have been exchanged (so ?*but1* instead of !*but1* and !*liq* instead of ?*liq*) is also deemed correct.

Question 7 **(10 points)**

Figure 4 shows an automaton, with initial state i_0.

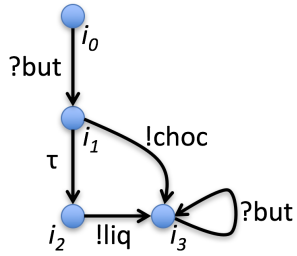


FIGURE 4 Input alphabet $L_I = \{?but\}$. Output alphabet $L_U = \{!liq, !choc\}$

(a) What must be filled in at the dots?

1. i_0 **after** ?but $= \ldots$
2. i_0 **after** ?but **refuses** $\{!liq\} = \ldots$
3. i_0 **after** ?but **refuses** $\{!choc\} = \ldots$

Solution:
1. i_0 **after** ?but $= \{i_1, i_2\}$
2. i_0 **after** ?but **refuses** $\{!liq\} = \mathit{false}$
3. i_0 **after** ?but **refuses** $\{!choc\} = \mathit{true}$

(b) Provide a trace that contains an output.

Solution: ?*but*·!*choc*

(c) Provide a suspension trace that is not a trace.

Solution: δ

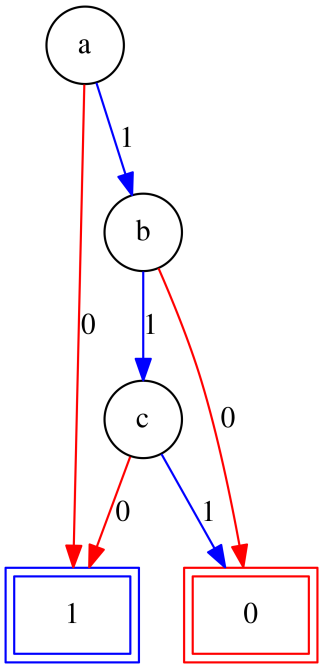
Question 8 **(10 points)**

Let f_0 and f_1 be the formulae:

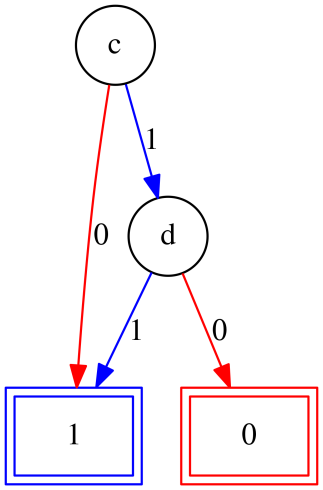
$$f_0 = a \rightarrow (b \wedge \neg c)$$
$$f_1 = d \vee \neg c$$

(a) Provide ROBDDs for both formulae, under variable ordering a, b, c, d.

Solution: The BDD for f_0 is:

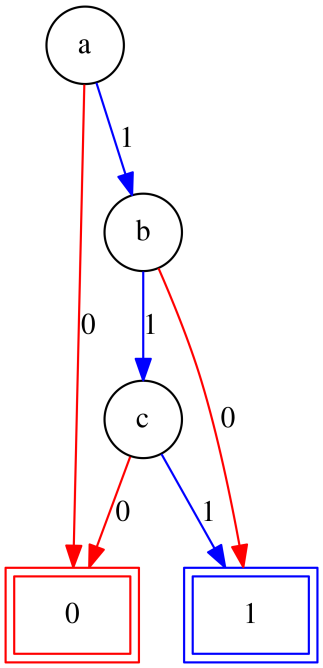


For f_1 the BDD is:

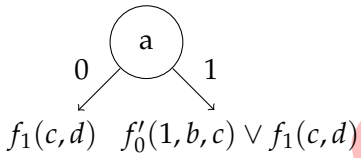


(b) Using Apply and Shannon's expansion to compute a disjunction of two ROBDDS, provide an ROBDD for formula $f_0 \rightarrow f_1$, under variable ordering a, b, c, d.

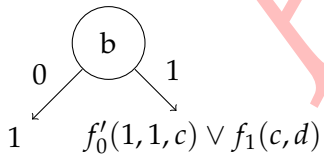
Solution: The key observation here is that $f_0 \rightarrow f_1$ is equivalent to $\neg f_0 \vee f_1$. Furthermore, given the BDD for f_0, the BDD for $f_0' = \neg f_0$ is obtained by negating the terminals (so 0 and 1 are exchanged). The BDD for $\neg f_0$ is thus



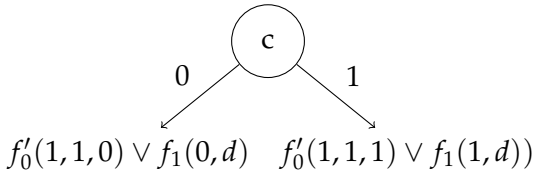
We use Apply and Shannon's expansion to compute the disjunction. We compare the roots, $a < c$, so we get $f_0'(a,b,c) \vee f_1(c,d) = (\neg a \wedge (f_0'(0,b,c) \vee f_1(c,d)) \vee (a \wedge (f_0'(0,b,c) \vee f_1(c,d))$. Since $f_0'(0,b,c) = 0$, the first branch simplifies to $f_1(c,d)$. We obtain the following start of our BDD.



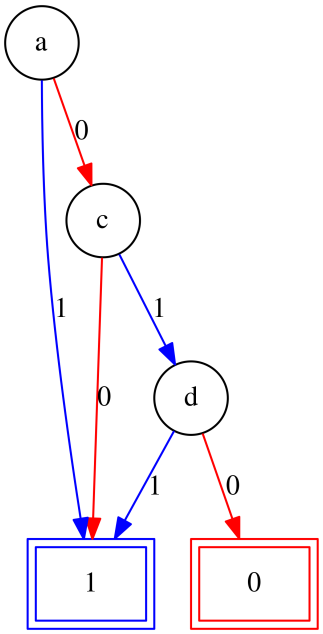
The computation for the first argument terminates, and we can immediately substitute the BDD for f_1. We only need to continue computing the BDD for the second argument, $f_0'(1,b,c) \vee f_1(c,d)$. Since $b < c$, we get $(\neg b \wedge (f_0'(1,0,c) \vee f_1(c,d))) \vee (b \wedge (f_0'(1,1,c) \vee f_1(c,d)))$. Since $f_0'(1,0,c)$, the first disjunct reduces to 1. The second disjunct cannot be simplified. This results in the following BDD node



We continue with $f_0'(1,1,c) \vee f_1(c,d)$, and expand c, so we get $(\neg c \wedge (f_0'(1,1,0) \vee f_1(0,d))) \vee (\neg c \wedge (f_0'(1,1,1) \vee f_1(1,d)))$. We obtain



Both arguments reduce to 1, so this sub-BDD can be simplified to 1. Then, in turn, both arguments to the b node lead to 1, and that BDD is also simplified to 1.
The minimal BDD that is thus obtained is



Question 9 **(15 points)**

Figure 5 shows a Kripke structure. State labels are triples indicating the values of atomic propositions x, y and z respectively. For example, state label $(0,1,0)$ indicates that x and z are true, and y is false.

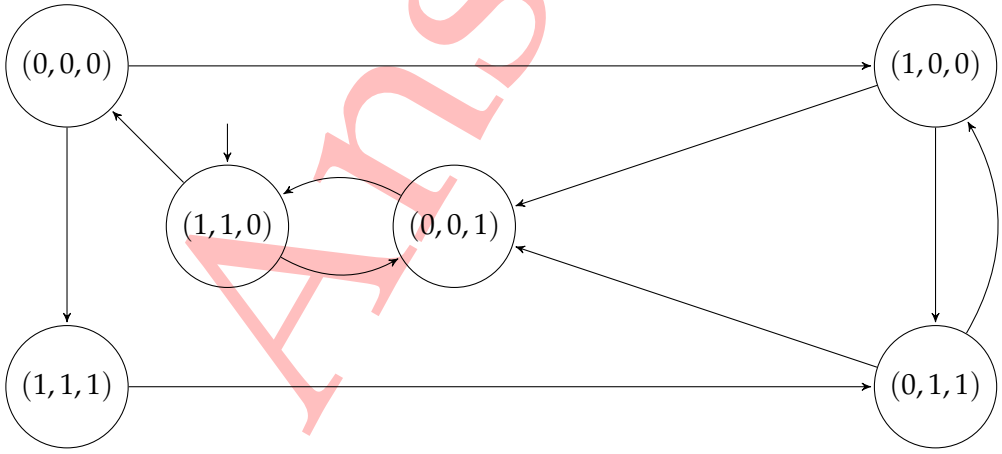
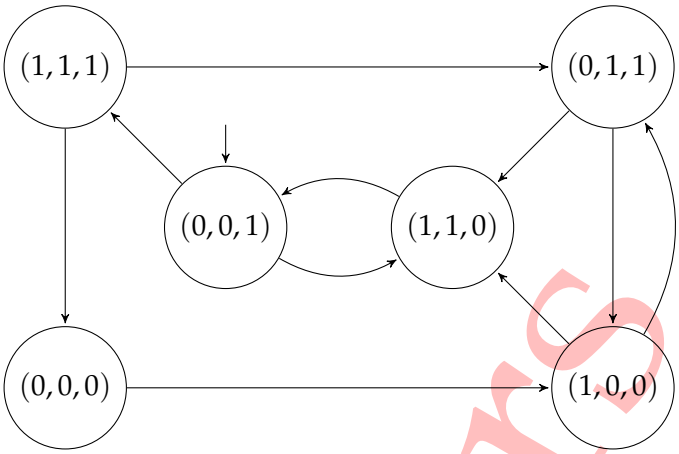


FIGURE 5 Kripke structure

(a) Provide a minimal BDD that symbolically represents the set of the four outer states.

Solution: The assignment text maps true to 0 and false to 1. This is a strange decision, and in grading both this mapping and the obvious mapping of true to 1 and false to 0

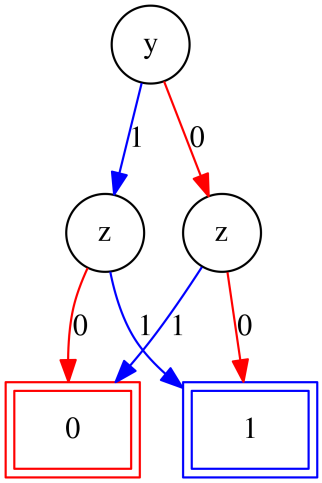
have been fully accepted. Let us first take a look at the mapping as proposed above. For ease of reading, we first modify the labelling of the states so that true and false coincide with 1 and 0.



The outer four states are then represented by the formula

$$(x \wedge y \wedge z) \vee (\neg x \wedge y \wedge z) \vee (\neg x \wedge \neg y \wedge \neg z) \vee (x \wedge \neg y \wedge \neg z)$$

Observe that in this formula the value for x is immaterial, so the formula can be simplified to $(y \wedge z) \vee (\neg y \wedge \neg z)$. For the size of the BDD the variable order does not matter. The minimal (RO)BDD we obtain is the following



(b) For each of the following formulae:
Apply the algorithm for symbolical model checking. This entails showing the evolution of set Z for each step of the algorithm. Does the formula hold for the Kripke structure?
- $\mathbf{AF}(x + y + z = 2)$
- $\mathbf{EG}(\neg z \rightarrow (x + y = 1))$

Solution: We base the solution on the Kripke structure provided in part a (with the relabelling).
We first compute $\mathbf{AF}(x + y + z = 2)$. Observe that $\mathbf{AF}(x + y + z = 2) = \mu Z.(x + y +$

$z = 2) \vee \mathbf{AX}(Z)$. The predicate transformer we use in the application of the symbolic algorithm is $\tau(Z) = (x + y + z = 2) \vee \mathbf{AX}(Z)$.
We are dealing with a least fixed point, so we start approximating the solution with *false*. The iterations are as follows:

$$
\begin{aligned}
Z_0 &= \mathit{false} \\
Z_1 &= \tau(Z_0) = \tau(\mathit{false}) = x + y + z = 2 \vee \mathbf{AX}(\mathit{false}) \\
&= \{(1,1,0),(0,1,1)\} \\
Z_2 &= \tau(Z_1) = \tau(\{(1,1,0),(0,1,1)\}) = x + y + z = 2 \vee \mathbf{AX}(\{(1,1,0),(0,1,1)\}) \\
&= \{(1,1,0),(0,1,1),(1,1,1),(1,0,0)\} \\
Z_3 &= \tau(Z_2) = \ldots \\
&= \{(1,1,0),(0,1,1),(1,1,1),(1,0,0),(0,0,0),(0,0,1)\} = S
\end{aligned}
$$

Since the full set S satisfies the property, the computation already terminates after Z_3. Since the intial state is included in the result, the Kripke structure satisfies the property.
Next, consider $\mathbf{EG}(\neg z \rightarrow (x + y = 1))$. We first obtain $\mathbf{EG}(\neg z \rightarrow (x + y = 1)) = \nu Z.(\neg z \rightarrow (x + y = 1)) \wedge \mathbf{EX}(Z)$. The corresponding predicate transformer is $\tau(Z) = (\neg z \rightarrow (x + y = 1)) \wedge \mathbf{EX}(Z)$. Since this is a greatest fixed point, we start the approximation from *true*.

$$
\begin{aligned}
Z_0 &= \mathit{true} \\
Z_1 &= \tau(Z_0) = (\neg z \rightarrow (z + y = 1)) \wedge \mathbf{EX}(Z_0) = \neg z \rightarrow (x + y = 1)) \\
&= \{(1,1,1),(0,1,1),(1,0,0),(0,0,1)\} \\
Z_2 &= \tau(Z_1) = \ldots \\
&= \{(1,1,1),(0,1,1),(1,0,0),(0,0,1)\} = Z_1
\end{aligned}
$$

So, the computation is stable, and the property is satisfied by these three states. The initial state is included, so the Kripke structure satisfies the property.

Solution: When we perform the computation on the original Kripke structure (without applying any relabelling) the computations become as follows.
We first compute $\mathbf{AF}(x + y + z = 2)$. Observe that $\mathbf{AF}(x + y + z = 2) = \mu Z.(x + y + z = 2) \vee \mathbf{AX}(Z)$. The predicate transformer we use in the application of the symbolic algorithm is $\tau(Z) = (x + y + z = 2) \vee \mathbf{AX}(Z)$.
We are dealing with a least fixed point, so we start approximating the solution with

false. The iterations are as follows:

$$
\begin{aligned}
Z_0 &= \mathit{false} \\
Z_1 &= \tau(Z_0) = \tau(\mathit{false}) = x + y + z = 2 \vee \mathbf{AX}(\mathit{false}) \\
&= \{(1,1,0),(0,1,1)\} \\
Z_2 &= \tau(Z_1) = \tau(\{(1,1,0),(0,1,1)\}) = x + y + z = 2 \vee \mathbf{AX}(\{(1,1,0),(0,1,1)\}) \\
&= \{(1,1,0),(0,1,1),(0,0,1),(1,1,1)\} \\
Z_3 &= \tau(Z_2) = \ldots \\
&= \{(1,1,0),(0,1,1),(0,0,1),(1,1,1),(1,0,0)\} \\
Z_4 &= \tau(Z_3) = \ldots \\
&= \{(1,1,0),(0,1,1),(0,0,1),(1,1,1),(1,0,0),(0,0,0)\} = S
\end{aligned}
$$

Since the full set S satisfies the property, the computation already terminates after Z_4. Since the intial state is included in the result, the Kripke structure satisfies the property.
Next, consider $\mathbf{EG}(\neg z \rightarrow (x + y = 1))$. We first obtain $\mathbf{EG}(\neg z \rightarrow (x + y = 1)) = \nu Z.(\neg z \rightarrow (x + y = 1)) \wedge \mathbf{EX}(Z)$. The corresponding predicate transformer is $\tau(Z) = (\neg z \rightarrow (x + y = 1)) \wedge \mathbf{EX}(Z)$. Since this is a greatest fixed point, we start the approximation from *true*.

$$
\begin{aligned}
Z_0 &= \mathit{true} \\
Z_1 &= \tau(Z_0) = (\neg z \rightarrow (z + y = 1)) \wedge \mathbf{EX}(Z_0) = \neg z \rightarrow (x + y = 1)) \\
&= \{(1,1,1),(0,1,1),(1,0,0),(0,0,1)\} \\
Z_2 &= \tau(Z_1) = \ldots \\
&= \{(1,1,1),(0,1,1),(1,0,0)\} \\
Z_3 &= \tau(Z_2) = \ldots \\
&= \{(1,1,1),(0,1,1),(1,0,0)\} = Z_2
\end{aligned}
$$

So, the computation is stable, and the property is satisfied by these three states. The initial state is included, so the Kripke structure satisfies the property.