VOORLOPIG ANTWOORDMODEL

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.*

You may answer the questions in English or in Dutch, and you can answer the questions in any order.

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, rounded.

Question 1: Kripke structures **(10 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 and 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 `either/do`-construct executes nondeterministically one of the blocks, and the `while`-loop is repeated until the given condition is false. Thus: each time the loop is entered, it may differ which block is executed.

(3 points) (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

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

Continue on the next page…

Next, we give the translation

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

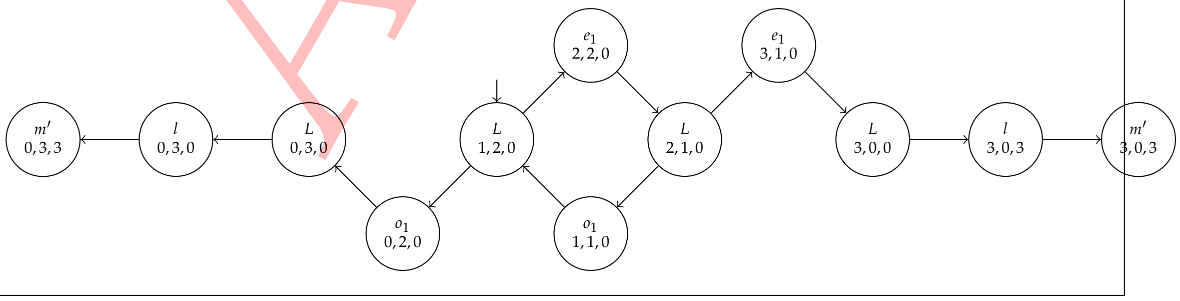
this is defined as the disjunction of the following four formulas:
- $pc = l \wedge pc' = l_1 \wedge same(V)$
- $pc = l \wedge pc' = l_2 \wedge same(V)$
- $\mathcal{C}(\ell_1, P_1, l')$
- $\mathcal{C}(\ell_2, P_2, l')$

(7 points) (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 and 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
   end while;
l:   z := x + y
m':
```

Next, we build a Kripke structure. Note that the states with labels `m`, `w`, `e0` and `o0` 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



Continue on the next page...

Question 2: Timed automata **(20 points)**

Consider the following timed automaton in UPPAAL's notation.

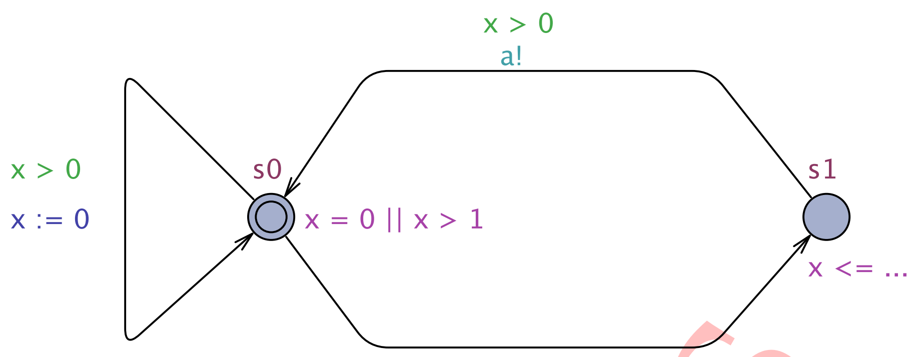


FIGURE 1 Timed automaton. Note that we use := for an assignment in an update, and the = sign for equality in guards and invariants.

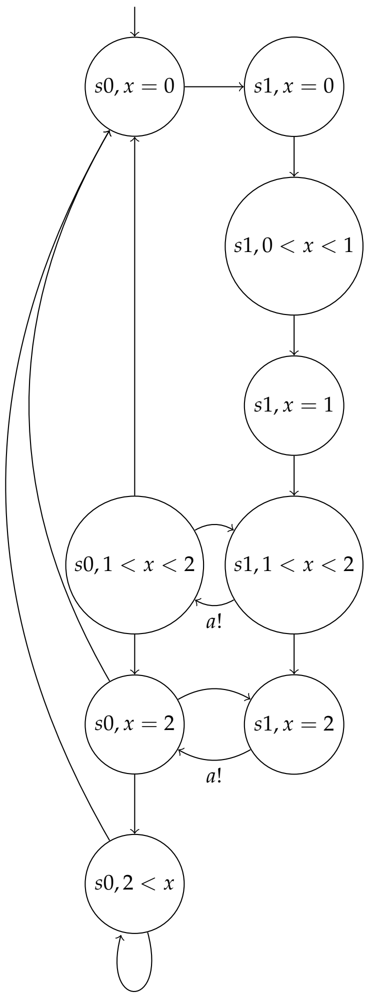
(3 points) (a) What is the lowest natural number that can be filled in at the dots (...), such that the automaton does not have a deadlock? Argue your answer.

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 $s0$.

(5 points) (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.

Continue on the next page...



(2 points) (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 $(s1, x = 2)$.

(10 points) (d) Assume the timed automaton from Figure 1 and the timed automaton shown below in Figure 2 synchronise (on the a action). Provide the parallel composition of these two automata (assume the dots are replaced by your value obtained in Question 2 (a); if you did not find a value, you may use 5).

Continue on the next page...

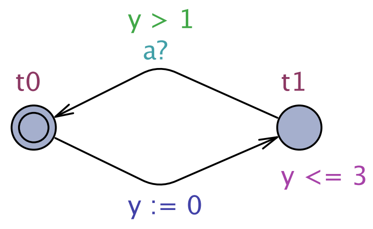
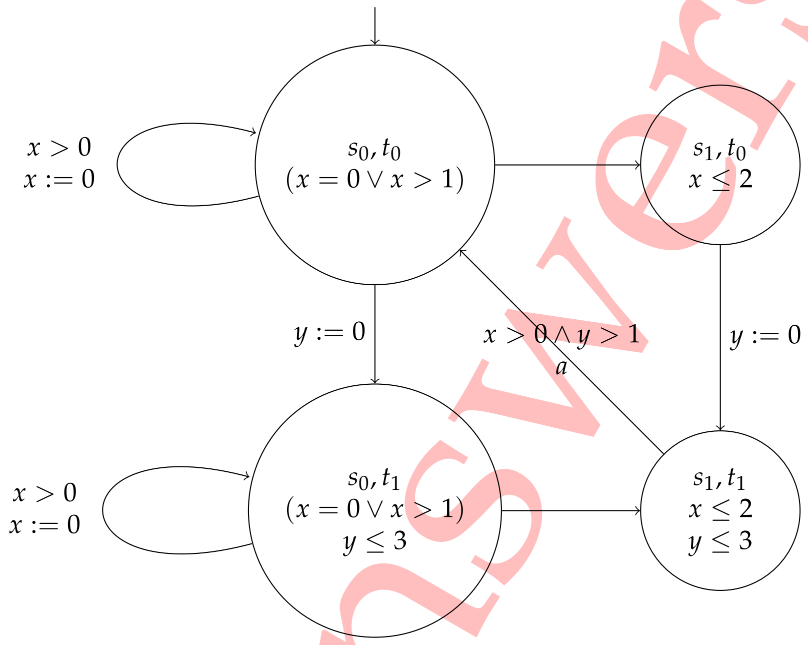


FIGURE 2 Timed automaton T. Note that we use $:=$ for an assignment in an update, and the $=$ sign for equality in guards and invariants.

Solution: Note that we give the solution in general timed-automata syntax instead of UPPAAL syntax. The meaning is the same.



Continue on the next page...

Question 3: Temporal logics **(15 points)**

(3 points) (a) Consider the Kripke structure from your answer in Question 1 (b). For this Kripke structure, 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. An example of a correct invariant is $AG(z = 0 \vee z = 3)$.

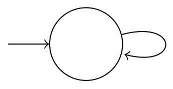
(2 points) (b) Again consider the Kripke structure from your answer in Question 1 (b). Does the property $\mathbf{AGAF}(z > 0)$ hold? Argue your answer.

Solution: If we look at the Kripke structure, we can observe there is a loop $L, (1, 2, 0) \rightarrow e_1, (2, 2, 0) \rightarrow L, (2, 1, 0) \rightarrow o_1, (1, 1, 0) \rightarrow L, (1, 2, 0) \rightarrow \cdots$ along which $z > 0$ never holds. This is a counterexample to $\mathbf{AGAF}(z > 0)$, and the property hence does not hold.

(5 points) (c) 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. If no such Kripke structure exists, argue why not.
- $\mathbf{A}(p\ \mathbf{U}\ q)$
- $p \rightsquigarrow q$

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

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



For the remainder of this question, 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.

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

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

(3 points) (e) 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 4: Explicit model checking **(10 points)**

Figure 3 shows a Kripke structure.

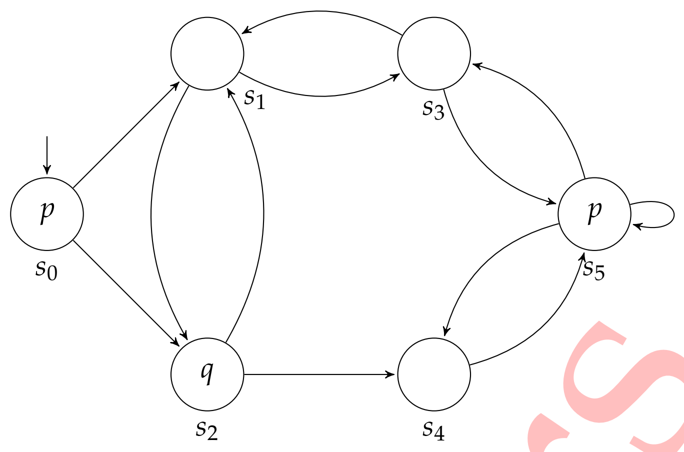


FIGURE 3 Kripke structure

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.

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

Since $s_0 \in S(\mathbf{EG}(\neg\mathbf{EG}(\neg p) \vee q))$, the Kripke structure satisfies the property $\mathbf{EG}(\mathbf{AF}(p) \vee q)$

Continue on the next page…

Question 5: Symbolic model checking **(20 points)**

(5 points) (a) Let f_0 and f_1 be the formulae:

$$\begin{aligned} f_0 &= a \rightarrow (b \wedge \neg c) \\ f_1 &= d \vee \neg c \end{aligned}$$

The ROBDDs for both formulae, under variable ordering a, b, c, d are shown in Figure 4. Provide an ROBDD for formula $f_0 \rightarrow f_1$, under variable ordering a, b, c, d. Show how you use Apply and Shannon's expansion to compute the resulting ROBDD.

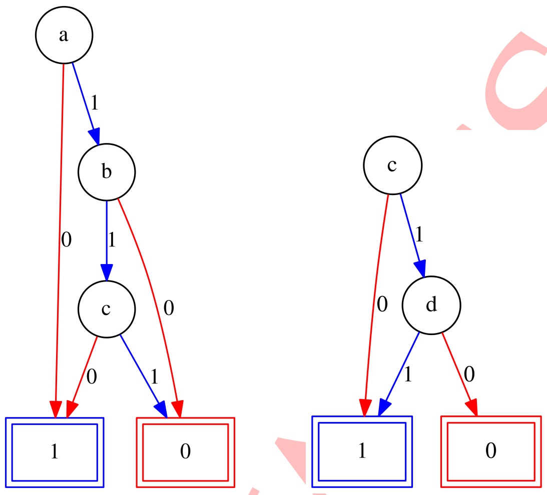
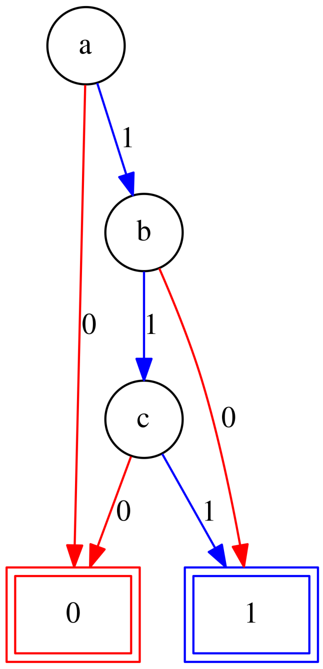


FIGURE 4 The ROBDDs for f_0 (left) and f_1 (right).

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

Continue on the next page…



We use Apply and Shannon's expansion to compute the disjunction. We compare the roots, $a < c$, so we get $f'_0(a,b,c) \lor f_1(c,d) = (\neg a \land (f'_0(0,b,c) \lor f_1(c,d)) \lor (a \land (f'_0(0,b,c) \lor f_1(c,d))$. Since $f'_0(0,b,c) = 0$, the first branch simplifies to $f_1(c,d)$. We obtain the following start of our BDD.

$$\begin{tikzcd}
& a \arrow[dl, "0"'] \arrow[dr, "1"] & \\
f_1(c,d) & & f'_0(1,b,c) \lor f_1(c,d)
\end{tikzcd}$$

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, $f'_0(1,b,c) \lor f_1(c,d)$. Since $b < c$, we get $(\neg b \land (f'_0(1,0,c) \lor f_1(c,d))) \lor (b \land (f'_0(1,1,c) \lor f_1(c,d)))$. 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

$$\begin{tikzcd}
& b \arrow[dl, "0"'] \arrow[dr, "1"] & \\
1 & & f'_0(1,1,c) \lor f_1(c,d)
\end{tikzcd}$$

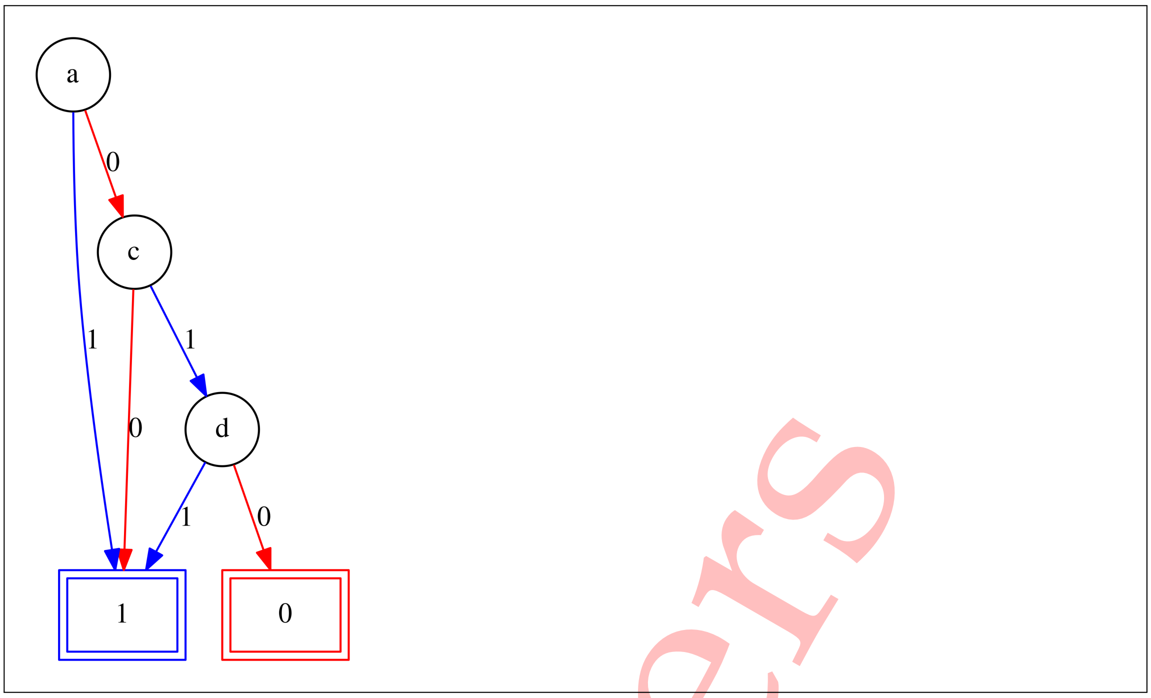
We continue with $f'_0(1,1,c) \lor f_1(c,d)$, and expand c, so we get $(\neg c \land (f'_0(1,1,0) \lor f_1(0,d))) \lor (c \land (f'_0(1,1,1) \lor f_1(1,d)))$. We obtain

$$\begin{tikzcd}
& c \arrow[dl, "0"'] \arrow[dr, "1"] & \\
f'_0(1,1,0) \lor f_1(0,d) & & f'_0(1,1,1) \lor f_1(1,d))
\end{tikzcd}$$

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

Continue on the next page…



Continue on the next page...

For the remainder of this question, consider the Kripke structure shown in Figure 5. In this figure, 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 false, and y is true.

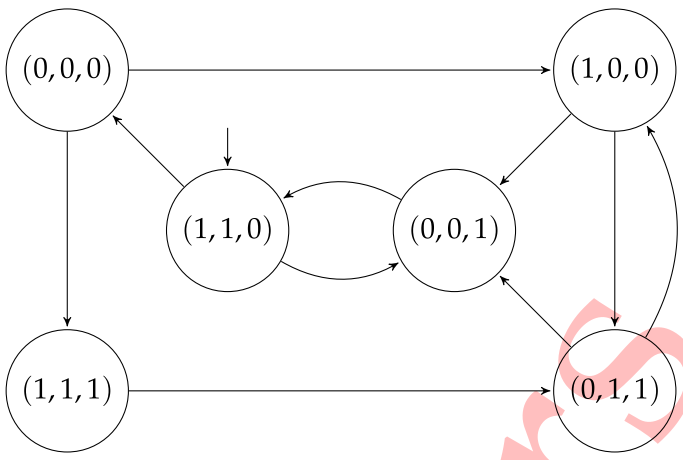


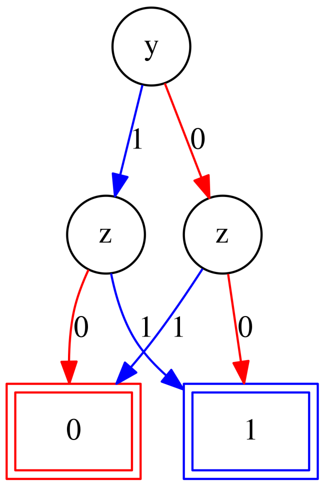
FIGURE 5 Kripke structure

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

Solution: The outer four states are represented by the formula

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

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



(10 points) (c) Apply the algorithm for symbolic model checking to the formula $\mathbf{AF}(x + y + z = 2)$. This entails showing the evolution of set Z for each step of the algorithm. Does the formula hold for the Kripke structure?

Continue on the next page...

Solution: 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:

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

Since the full set S satisfies the property, the computation already terminates after Z_4. Since the initial state is included in the result, the Kripke structure satisfies the property.

Continue on the next page…

Question 6: Labelled transition systems **(10 points)**

Figure 6 shows an IO automaton with initial state i_0.

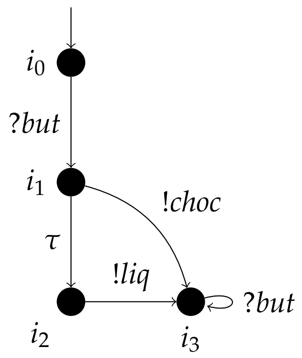


FIGURE 6 Input alphabet $L_I = \{?\text{but}\}$. Output alphabet $L_U = \{!\text{liq}, !\text{choc}\}$

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

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

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

(2 points) (b) Provide a trace that contains an output.

Solution: $?\mathit{but}\cdot!\mathit{choc}$

(2 points) (c) Provide a suspension trace that is not a trace. If no such suspension trace exists, explain why not.

Solution: δ

(3 points) (d) Is the automaton input-enabled? Argue your answer.

Solution: The automaton is not input-enabled. Not in every state, all input actions are allowed. In particular, in i_2 no inputs are enabled.

Continue on the next page…

Question 7: IOCO **(5 points)**
Figure 7 shows two IO automata: an implementation i (left) and a specification s (right).

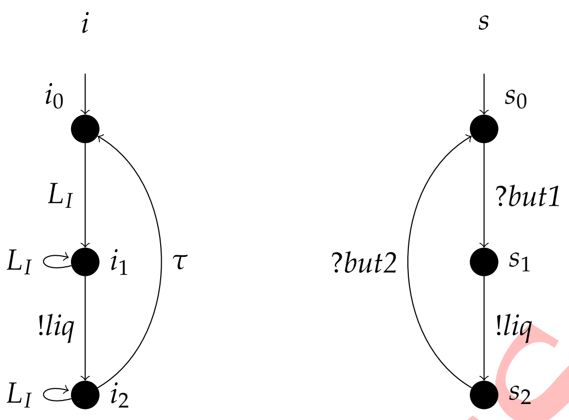


FIGURE 7 IO automata. Input alphabet $L_I = \{?but1, ?but2\}$. Output alphabet $L_U = \{!liq\}$

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 $\sigma = ?but1 \cdot !liq \cdot ?but2$.

We get the following after sets:
- $i\ \textbf{after}\ \sigma = \{i_0, i_1, i_2\}$ (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).
- $s\ \textbf{after}\ \sigma = \{s_0\}$.

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

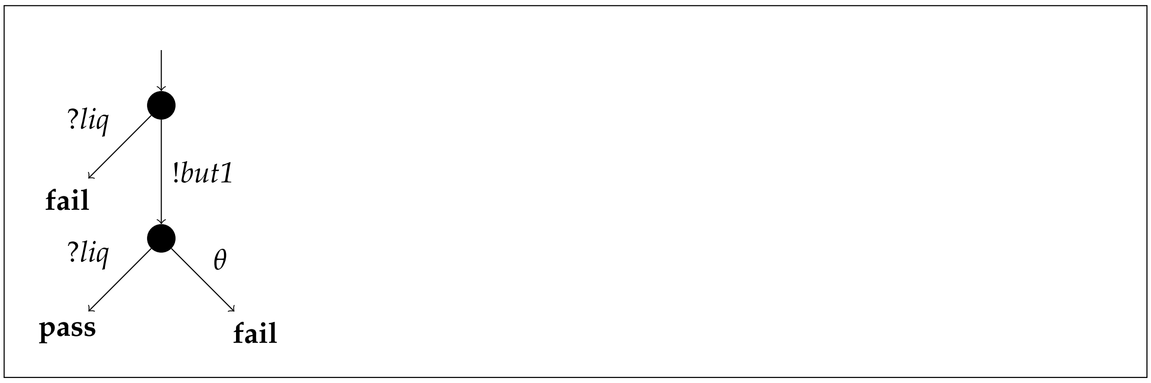
Hence, $out(i\ \textbf{after}\ \sigma) \not\subseteq out(s\ \textbf{after}\ \sigma)$, and thus not i **ioco** s.

Question 8: Test case generation & execution **(10 points)**
Consider the two IO automata from Figure 7: an implementation i (left) and a specification s (right).

(5 points) (a) 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:

Continue on the next page…



(2 points) (b) Is your test case sound? Argue your answer.

Solution: The test case is obtained using the construction described by Tretmans. According to Theorem 2.1 on page 34 of Tretmans' paper, the algorithm only generates sound test cases.

(3 points) (c) Is the test suite consisting of only the test case defined in Question 8 (a) exhaustive? Argue your answer.

Solution: No, the test suite is not exhaustive. This can be seen, for example, as follows. The implementation from Figure 7 passes the test suite, yet we have seen that the implementation does not conform to the specification. Since the implementation is not **ioco**-conforming to the specification, there must be at least one failing test case in the test suite according to Definition 17.3 in Tretmans' paper.