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. 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: Kripke structures **(10 points)**

Consider the following program code. In the initial state, variable x has some unknown value from 1 up to and including 4 and variable y has value 4.

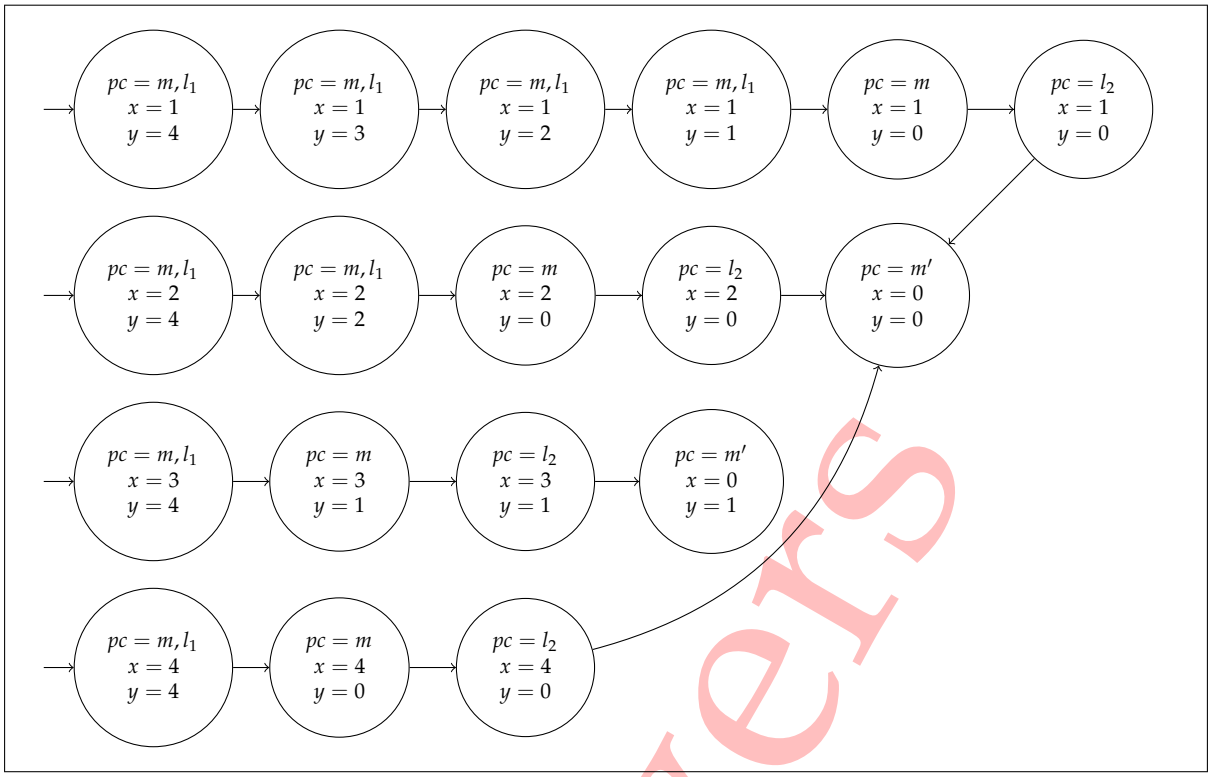
```
while (x <= y && y > 0)
do
    y := y - x
end while
x := 0
```

Provide a Kripke structure modelling this program code.

Solution: We first model the program counter using the labels m, l_1, l_2, m'.

```
m:   while (x <= y && y > 0)
     do
l1:      y := y - x
      end while
l2:   x := 0
```

`m'`: Next we translate the program in a rather coarse way; strictly following the translation to Kripke structure results in a Kripke structure with way more states. In particular, we merge states m and l_1 whenever the guard is true.



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

Figure 1 shows a timed automaton in UPPAAL's notation.

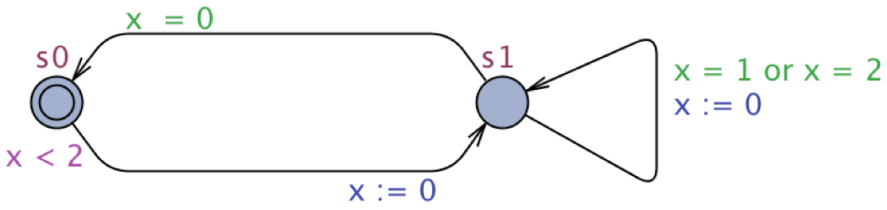
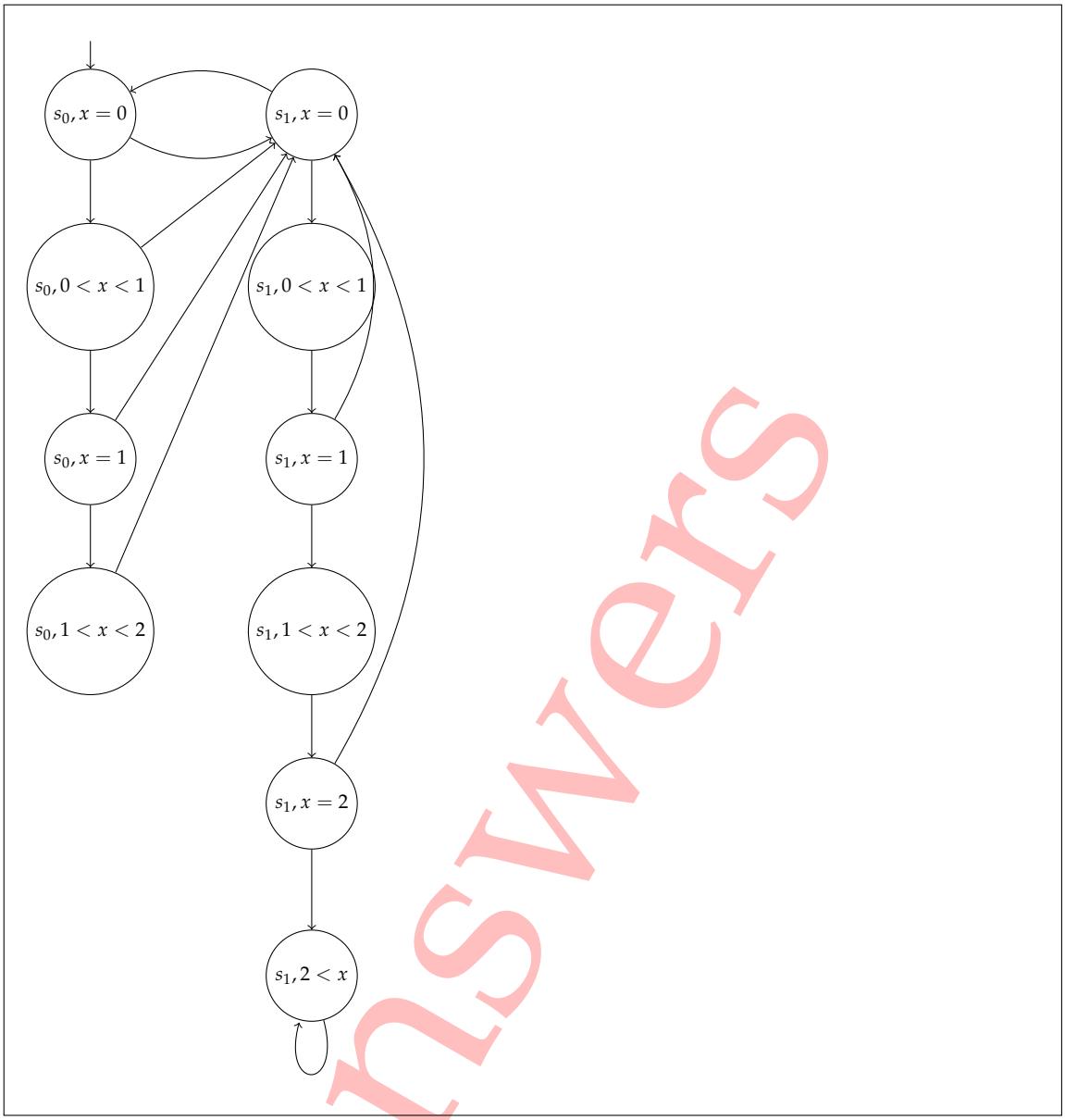


FIGURE 1 Timed automaton (UPPAAL notation). Note that we use := for an assignment in an update, and the = sign for equality in guards.

(5 points) (a) Draw the corresponding region automaton.

Solution: Observe that the largest constant in the automaton is 2. We thus obtain the following region automaton.



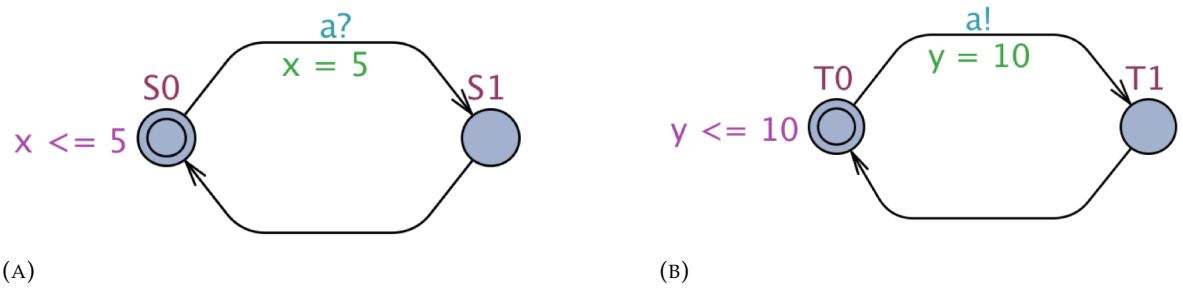
(3 points) (b) Does the UPPAAL automaton have a deadlock? If yes, then provide a trace leading to that deadlock. If not, then argue why not. In both cases, use the region automata in your argument.

Solution: The automaton does not have a deadlock. If we look at the region automaton, all states have an outgoing transition, so in every state either a transition is possible, or time is allowed to pass.

(2 points) (c) Does the UPPAAL automaton exhibit Zeno behavior? Argue your answer. Use the region automaton in your argument.

Solution: Yes, the automaton exhibits Zeno behaviour. Between $(s_0, x = 0)$ and $(s_1, x = 0)$ an infinite number of transitions can happen without letting time pass.

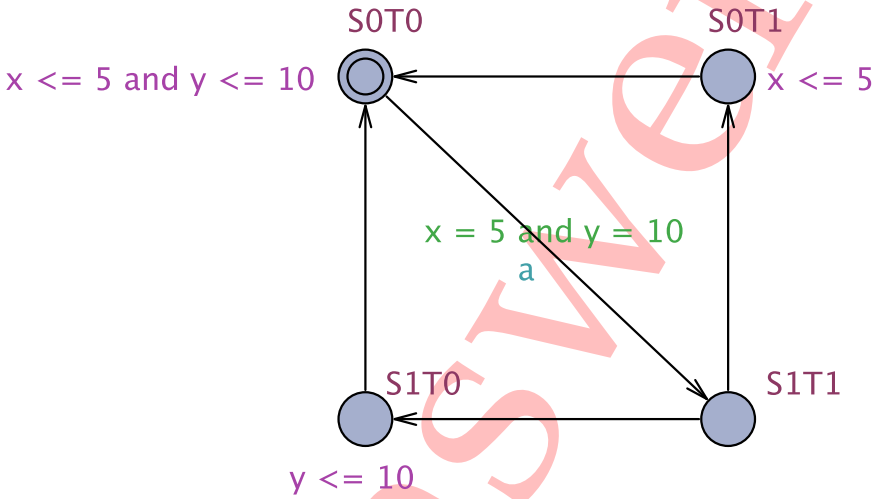
(10 points) (d) Figure 2 shows two UPPAAL automata. Provide the parallel composition of these two automata.



(A) (B)

FIGURE 2 UPPAAL automata.

Solution: The composition of the automata is the following.



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

(2 points) (a) Recall the timed automaton from Figure 1. Does the following CTL formula hold? Argue your answer. You may use the region automaton you constructed in Question 2 (a) in your answer.

$$\mathbf{AG}(s_1 \rightarrow \mathbf{AF}(s_0))$$

Solution: No, this formula does not hold, since whenever we have reached $(s_1, 2 < x)$ there is no way to get back to s_0. This can clearly be seen from the region automaton constructed in Question 2 (a).

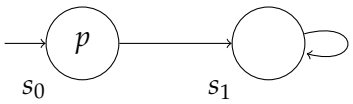
(3 points) (b) Consider the Kripke structure you constructed in Question 1. For each of the following two CTL formulae, state whether the formula holds for your Kripke structure. Argue your answer. Note that $\rightsquigarrow$ is UPPAAL's liveness operator and $\rightarrow$ is normal logical implication.
– $x \in \{0, 2, 4\} \rightsquigarrow (x = 0 \wedge y = 0)$

– $\mathbf{EG}(x \in \{0,2,4\} \rightarrow y \in \{0,2,4\})$

Solution:
– $x \in \{0,2,4\} \rightsquigarrow (x = 0 \wedge y = 0)$ does not hold. We are dealing with a liveness property that says that whenever $x \in \{0,2,4\}$ holds always eventually $x = 0 \wedge y = 0$ holds. In particular, this should hold in the state $pc = m'$, $x = 0$. $y = 1$, from which the desired situation $x = 0$ and $y = 0$ cannot be reached.
– $\mathbf{EG}(x \in \{0,2,4\} \rightarrow y \in \{0,2,4\})$ holds; a witness is the path starting in the initial state $(pc = m, l_1, x = 2, y = 4)$.

(10 points) (c) Provide a Kripke structure that demonstrates the difference between the CTL formulae $\mathbf{AF}(p)$ and $\mathbf{AG}\ \mathbf{AF}(p)$. Thus, the Kripke structure must be true for *exactly* one of these CTL formulae.

Solution: The following Kripke structure satisfies $\mathbf{AF}(p)$ but not $\mathbf{AG}\ \mathbf{AF}(p)$.



Question 4: Explicit model checking **(10 points)**

Figure 3 shows a Kripke structure. Execute the algorithm for explicit state model checking step-by-step for the following CTL formulae. This entails showing the steps performed, and providing the output of the algorithm.

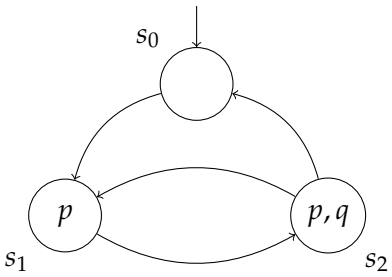


FIGURE 3 Kripke structure

(5 points) (a) $s_0 \models \mathbf{EF}\,\mathbf{AG}(p)$

Solution: First, observe that the explicit state model checking algorithm is defined for the subset of CTL containing atomic propositions, and the operators $\neg$, $\vee$, **EX**, **EG**, and **EU**. We therefore first rewrite the formula as follows: $\mathbf{EF}\,\mathbf{AG}\,(p) \equiv \mathbf{E}\,(\text{true}\,\mathbf{U}\,\mathbf{AG}\,(p)) \equiv \mathbf{E}\,(\text{true}\,\mathbf{U}\,\neg\mathbf{EF}\,(\neg p)) \equiv \mathbf{E}\,(\text{true}\,\mathbf{U}\,\neg\mathbf{E}\,(\text{true}\,\mathbf{U}\,\neg p))$. We then stepwise label the nodes, starting with the smallest subformulas. We obtain the following labellings.

$$\begin{aligned} S(p) &= \{s_1, s_2\} \\ S(\neg p) &= \{s_0\} \\ S(\mathbf{E}\,(\text{true}\,\mathbf{U}\,\neg p)) &= \{s_0, s_1, s_2\} \\ S(\neg\mathbf{E}\,(\text{true}\,\mathbf{U}\,\neg p)) &= \emptyset \\ S(\mathbf{E}\,(\text{true}\,\mathbf{U}\,\neg\mathbf{E}\,(\text{true}\,\mathbf{U}\,\neg p))) &= \emptyset \end{aligned}$$

So, $\mathbf{EF}\,\mathbf{AG}(p)$ is false for this Kripke structure.

(5 points) (b) $s_0 \models \mathbf{AF}\,\mathbf{EG}\,p$

Solution: Again, we first rewrite the property to the subset of CTL containing atomic propositions, and the operators $\neg$, $\vee$, **EX**, **EG**, and **EU**. We get $\mathbf{AF}\,\mathbf{EG}(p) \equiv \neg\mathbf{EG}(\neg\mathbf{EG}(p))$. Applying the labelling entails

$$\begin{aligned} S(p) &= \{s_1, s_2\} \\ S(\mathbf{EG}(p)) &= \{s_1, s_2\} \\ S(\neg\mathbf{EG}(p)) &= \{s_0\} \\ S(\mathbf{EG}(\neg\mathbf{EG}(p))) &= \emptyset \\ S(\neg\mathbf{EG}(\neg\mathbf{EG}(p))) &= \{s_0, s_1, s_2\} \end{aligned}$$

So, **AF EG**(p) is true for this Kripke structure.

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

Figure 4 shows a Kripke structure.

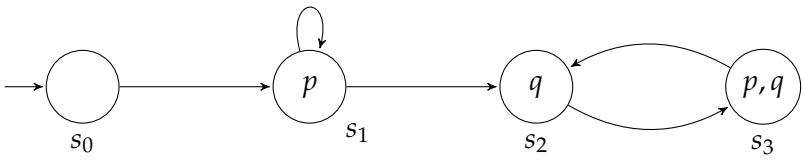
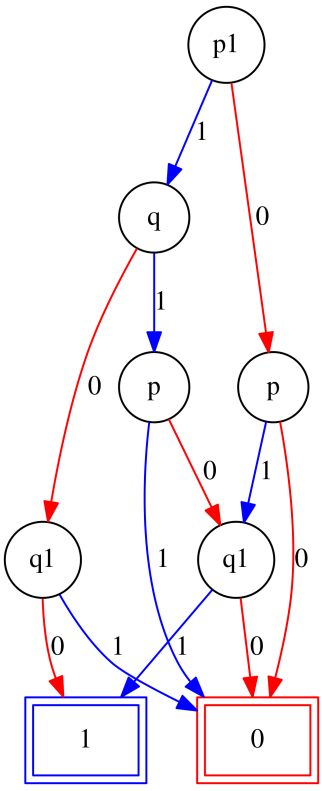


FIGURE 4 Kripke structure

(10 points) (a) Provide a BDD that symbolically represents the transition relation of the Kripke structure. Use variable ordering p', q, p, q'.

Solution: A BDD representing the transition relation is the following. Note that instead of p' and q' the new values are represented as $p1$ and $q1$ in this BDD.



For each of the following formulas, apply the algorithm for symbolic model checking to the Kripke structure from Figure 4, by showing the evolution of set Z for each step of the algorithm. You do not have to give the actual BDDs, but you can give the sets of states represented by the BDDs.

(5 points) (b) **EG**(p)

Solution: Observe that $\mathbf{EG}(p) = \nu Z.p \wedge \mathbf{EX}(Z)$. We therefore use the predicate transformer $\tau(Z) = p \wedge \mathbf{EX}(Z)$, and compute its greatest fixed point.

$$
\begin{aligned}
Z_0 &= \text{True} \\
Z_1 &= \{s_1, s_3\} \\
Z_2 &= \{s_1\} \\
Z_3 &= \{s_1\} = Z_2
\end{aligned}
$$

So, the computation is stable, and the property is satisfied only in s_1, hence the property does not hold for the Kripke structure.

(5 points) (c) $\mathbf{AF}(p \wedge q)$

Solution: Observe that $\mathbf{AF}(p \wedge q) = \mu Z.(p \wedge q) \vee \mathbf{AX}(Z)$. We therefore use the predicate transformer $\tau(Z) = (p \wedge q) \vee \mathbf{AX}(Z)$ and compute its least fixed point.

$$
\begin{aligned}
Z_0 &= \text{False} \\
Z_1 &= \{s_3\} \\
Z_2 &= \{s_3, s_2\} \\
Z_3 &= \{s_3, s_2\} = Z_2
\end{aligned}
$$

So, the computation is satisfied in s_2 and s_3, hence the property does not hold for the Kripke structure.

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

Figure 5 shows two IO automata: an implementation i and a specification s.

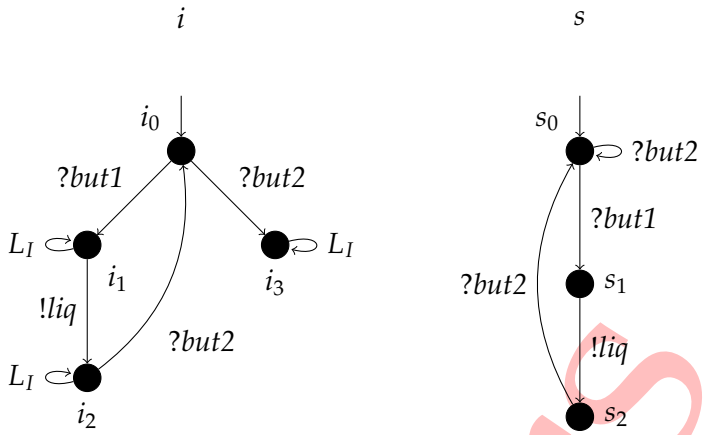


FIGURE 5 IO automata. Input alphabet $L_I = \{?but1, ?but2\}$. Output alphabet $L_U = \{!liq\}$

(2 points) (a) Is the implementation, i, deterministic? Argue your answer.

Solution: No, the implementation is not deterministic. From state i_2 there are two outgoing transitions labelled $?but2$.

(2 points) (b) Is the specification, s, input-enabled? Argue your answer.

Solution: No, the specification is not input-enabled. In state s_1, no input-action is enabled.

(2 points) (c) Consider the implementation i and specification s. Which states in these two LTSs are quiescent?

Solution:
- In the implementation i, states i_0, i_2 and i_3 are quiescent.
- In the specification s, states s_0 and s_2 are quiescent.

(2 points) (d) For each of the following predicates, state whether it is *true* or *false*.

(i) i_0 **after** $?but1$ **refuses** $\{!liq\}$
(ii) i_0 **after** $?but1$ **refuses** $\{!liq, \delta\}$
(iii) i_0 **after** $?but1 \cdot !liq$ **refuses** $\{!liq, \delta\}$
(iv) s_0 **after** $?but2 \cdot ?but1$ **refuses** $\{\delta\}$

Solution:
(i) i_0 **after** $?but1$ **refuses** $\{!liq\} = false$
(ii) i_0 **after** $?but1$ **refuses** $\{!liq, \delta\} = false$

(iii) i_0 **after** ?*but1* · !*liq* **refuses**$\{!liq, \delta\} = false$
(iv) s_0 **after** ?*but2* · ?*but1* **refuses**$\{\delta\} = true$

(2 points) (e) Give a suspension trace of the implementation that is not a trace. If no such suspension trace exists, argue why not.

Solution: Any suspension trace of the implementation that contains quiescence (δ) is not a trace. The simplest such trace, in this case, is δ.

Question 7: IOCO **(5 points)**

Consider implementation i and specification s from Figure 5. 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. Recall that i **ioco** $s \stackrel{def}{=} \forall \sigma \in Straces(s).\mathbf{out}(i \textbf{ after } \sigma) \subseteq \mathbf{out}(s \textbf{ after } \sigma)$. The trace ?but2·?but1 is a suspension trace of the specification, i.e., ?but2·?but1 $\in Straces(s)$. We have:
– $\mathbf{out}(i \textbf{ after } ?but2 \cdot ?but1) = \{\delta\}$
– $\mathbf{out}(s \textbf{ after } ?but2 \cdot ?but1) = \{!liq\}$

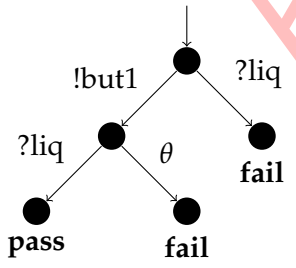
so, $\mathbf{out}(i \textbf{ after } ?but2 \cdot ?but1) \not\subseteq \mathbf{out}(s \textbf{ after } ?but2 \cdot ?but1)$, thus not i **ioco** s.

Question 8: Test case generation & execution **(10 points)**

Consider implementation i and specification s from Figure 5.

(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 smallest testcase we can construct that satisfies the additional requirements is the following.



(2 points) (b) Give a test run of your test case with implementation i.

Solution: A test run of the test case t with implementation i is a trace of $t \rceil\!\| i$ that ends in **pass** of **fail**. An example of such a test run is $but1 \cdot liq \cdot$ **pass**.

(3 points) (c) Argue that your test case is sound. What is the smallest change you can think of, that makes the test case unsound?

Solution: Recall that soundness means that for every test t generated, we have that t will never fail with a correct implementation. In other words if i **ioco** s then i **passes** t. The test case we have constructed is such, using the sound algorithm from Tretmans' paper.
A change that makes the testcase unsound should lead to a failure, even for an implementation that conforms to the specification. An example of such a change would be changing the **pass** in the testcase to **fail**.