Testing Techniques
Assignments & Exercises on *Model-Based Testing*

Jan Tretmans
TNO-ESI – Radboud University

October 21, 2014

1 Theory of Labelled Transition Systems

1.1 Testing equivalences

Consider the processes Q_1, Q_2, Q_3, Q_4, and Q_5, which are represented as labelled transition systems in Figure 1 with label set $L = \{a, b, c, d\}$.

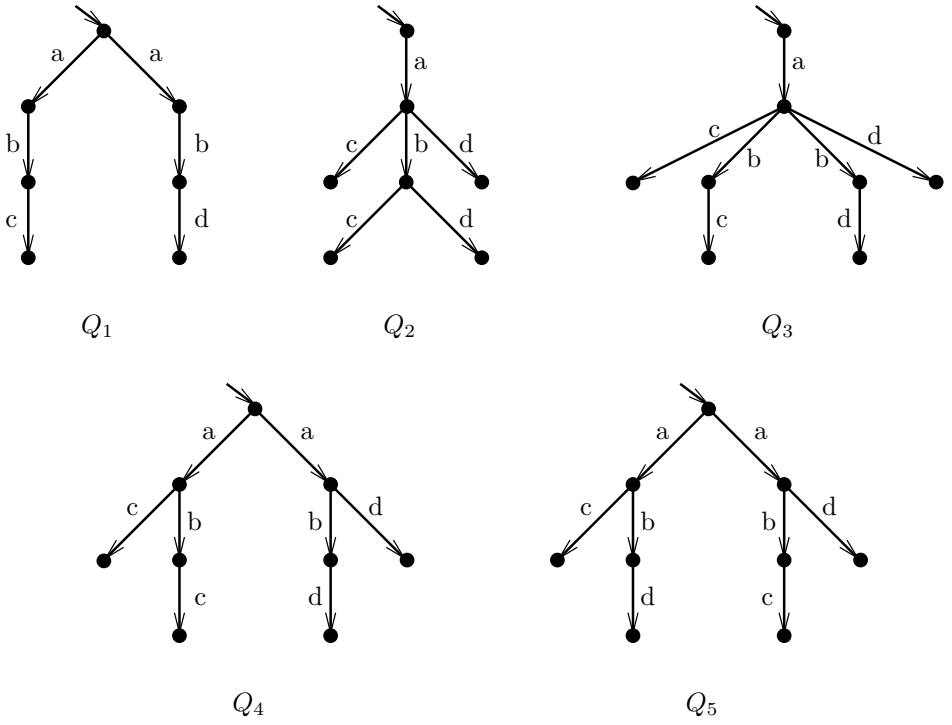


Figure 1:

a) Which pairs of processes are isomorphic, i.e., there is a 1:1 mapping between states and transitions? (Preferably, present your answers in a matrix.)

b) Which pairs of processes are trace equivalent $\approx_{tr}$?

c) Which pairs of processes are completed trace equivalent $\approx_{ct}$?

d) Describe test experiments, informally, which may be used to discriminate between the processes Q_1, Q_2, Q_3, Q_4, and Q_5.

e) Which pairs of processes are testing equivalent $\approx_{te}$?

f) Give an experiment to discriminate between processes Q_4 and Q_5. Do you think that Q_4 and Q_5 are testing equivalent $\approx_{te}$?

g) Which pairs of processes are refusal equivalent (= failure trace equivalent $\approx_{ft}$)?

h) Suppose you have an *undo*-action. Which processes can then be distinguished?

i) Now consider the processes as input-output transition systems with $L_I = \{a, b\}$ and $L_U = \{c, d\}$ and completed with self-loops in order to make them input-complete. Which pairs are **ioco**-related?

1.2 Testing equivalences

Consider the processes P_1, P_2, P_3 and P_4 which are represented as labelled transition systems in Figure 2 with labelset $L = \{a, b, c, d\}$.

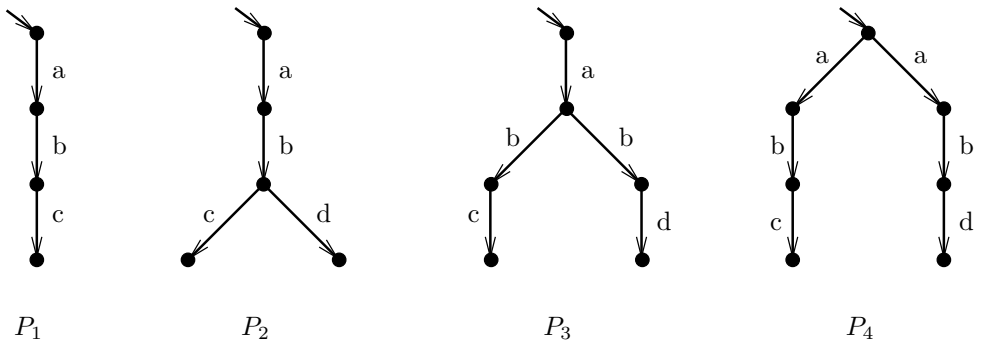


Figure 2:

a) Which pairs of processes are isomorphic, i.e., there is a 1:1 mapping between states and transitions?

b) Which pairs of processes are in trace preorder relation $\leq_{tr}$?

c) For P_2, P_3 and P_4: describe experiments, informally, which may lead to observing the differences between the processes.

d) Based on the answers in (c), which pairs of processes are testing equivalent $\approx_{te}$? (Or, equivalently, which pairs of processes can be distinguished based on tests consisting of labelled transition systems?)

e) Suppose that testers have an *undo*-action which allows to undo a previously taken transition, i.e., with *undo* we can go to the previous state. Which pairs of processes can be distinguished based on tests with *undo* actions?

f) Now consider the processes as input-output transition systems with $L_I = \{a, b\}$ and $L_U = \{c, d\}$. Modify the processes P_1, P_2, P_3 and P_4 such that they are indeed input-output transition systems.

g) Which pairs of processes, considered as input-output transition systems, are **ioco**-related?

1.3 Testing equivalences

Let $p, i, s \in \mathcal{LTS}(L)$ be labelled transition systems with actions in L, but without internal actions τ. The standard definitions for transitions are used for $p \xrightarrow{a} p'$ and $p \stackrel{\sigma}{\Longrightarrow} p'$, with $a \in L$ and $\sigma \in L^*$, respectively. Moreover, the following definitions from the lecture notes are used – where $A \in \mathcal{P}(L)$ (which is equivalent to $A \subseteq L$), and $\sigma \in L^*$:

1. $p \xrightarrow{A} p' \quad \Leftrightarrow_{\text{def}} \quad p = p' \;\text{ and }\; \forall a \in A :\; p \not\xrightarrow{a}$
2. $p \,\mathbf{after}\, \sigma \,\mathbf{refuses}\, A \quad \Leftrightarrow_{\text{def}} \quad \exists p' :\; p \stackrel{\sigma}{\Longrightarrow} p' \;\text{ and }\; \forall a \in A :\; p' \not\xrightarrow{a}$
3. $\mathit{traces}(p) \quad =_{\text{def}} \quad \{\, \sigma \in L^* \mid p \stackrel{\sigma}{\Longrightarrow} \,\}$
4. $\mathit{Ftraces}(p) \quad =_{\text{def}} \quad \{\, \varphi \in (L \cup \mathcal{P}(L))^* \mid p \stackrel{\varphi}{\Longrightarrow} \}$
   where $\stackrel{\varphi}{\Longrightarrow}$ is the extension of $\stackrel{\sigma}{\Longrightarrow}$ with labels in $\mathcal{P}(L)$
5. $i \leq_{tr} s \quad \Leftrightarrow_{\text{def}} \quad \mathit{traces}(i) \subseteq \mathit{traces}(s)$
6. $i \leq_{te} s \quad \Leftrightarrow_{\text{def}} \quad \forall \sigma \in L^*,\, \forall A \subseteq L :\;\; i \,\mathbf{after}\, \sigma \,\mathbf{refuses}\, A \;\text{ implies }\; s \,\mathbf{after}\, \sigma \,\mathbf{refuses}\, A$
7. $i \leq_{rf} s \quad \Leftrightarrow_{\text{def}} \quad \mathit{Ftraces}(i) \subseteq \mathit{Ftraces}(s)$

Prove the following properties, and indicate in each step which argument (definition, property) you use:

a) $\sigma \in \mathit{traces}(p)$ iff $p \,\mathbf{after}\, \sigma \,\mathbf{refuses}\, \emptyset$

b) $p \,\mathbf{after}\, \sigma \,\mathbf{refuses}\, A$ iff $\sigma{\cdot}A \in \mathit{Ftraces}(p)$

c) $i \leq_{te} s$ implies $i \leq_{tr} s$

d) $i \leq_{rf} s$ implies $i \leq_{te} s$

1.4 Testing equivalences

Given behaviour expression $P := Q \;|[b]|\; \mathbf{i}; R$, with $Q := a; (b; \mathbf{stop} \;|||\; Q)$ and $R := b; R$.

a) Draw the labelled transition system for P.

b) Give the behaviour expression of LTS P in Figure 3.

c) Does $P \sim Q$ hold? Show why or why not.

d) Does $P \approx Q$ hold? Show why or why not.

e) Does $Q \leq_{tr} P$ hold? Show why or why not.

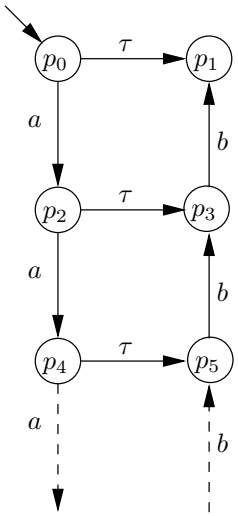
f) Does $P \leq_{tr} Q$ hold? Show why or why not.
Now let P' be the IOTS made out of LTS P by considering $a \in L_I$ and $b \in L_U$. In the same way let Q' be the IOTS made out of LTS Q by considering $a \in L_I$ and $b \in L_U$.

g) Does $Q \leq_{te} P$ hold? Show why or why not.

1.5 Testing equivalences

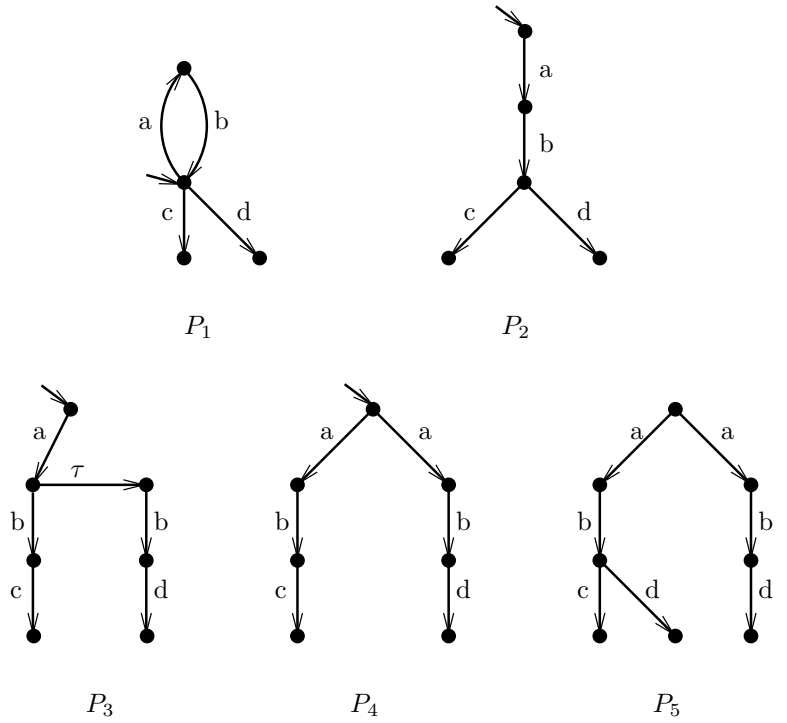
Consider the processes P_1, P_2, P_3, P_4 and P_5 which are represented as labelled transition systems in Figure 4 with labelset $L = \{a, b, c, d\}$.

a) Which pairs of processes are isomorphic, i.e., there is a 1:1 mapping between states and transitions?



P

Figure 3:



P_1 P_2 P_3 P_4 P_5

Figure 4:

b) Give the pairs of processes that are *not* trace equivalent $\not\approx_{tr}$, and show why.

c) Give the pairs of processes that are *not* completed trace equivalent $\not\approx_{ct}$, and show why.

d) Describe test experiments, informally, which may be used to discriminate between the processes P_1, P_2, P_3, P_4, and P_5.

e) Give the pairs of processes that are *not* testing equivalent $\not\approx_{te}$, and show why.

f) Give the pairs of processes that are *not* failure trace equivalent $\not\approx_{ft}$, and show why. (Failure trace equivalence is the same as refusal equivalence – at least as we do not have infinite sequences of τ-actions.)

g) Suppose you have an *undo*-action. Which processes can then be distinguished?

h) Now consider the processes as input-output transition systems with $L_I = \{a, b\}$ and $L_U = \{c, d\}$ and completed with self-loops in order to make them input-complete. Which pairs are **ioco**-related?

1.6 Properties of labelled transition systems

In the following exercises, let $p \in \mathcal{LTS}(L)$ be a (state of a) labelled transition system, and $\sigma, \sigma' \in L^*$.

a) In the following proof exert every step in detail, and indicate in each step which argument (definition, property, ...) you use.

Prove that:

$$p \textbf{ after } (\sigma \cdot \sigma') \;=\; (\, p \textbf{ after } \sigma \,) \textbf{ after } \sigma'$$

b) Proof that:

$$der(p) \textbf{ after } \sigma \;=\; der(p)$$

c) Describe in words what the following formula expresses:

$$\forall \sigma \in L^*. p \notin (\, p \textbf{ after } \sigma \,)$$

d) Describe in words what the following formula expresses:

$$\forall s \in der(p), \forall \sigma \in L^* :\; s \notin (\, s \textbf{ after } \sigma \,)$$

e) Proof that: p has finite behaviour implies $\forall s \in der(p), \forall \sigma \in L^* :\; s \notin (\, s \textbf{ after } \sigma \,)$.

f) Give an example that shows: p has finite behaviour does *not* imply that p is finite state.

g) Give an example that shows: p has finite behaviour does *not* imply that p is deterministic.

h) Give an example that shows: p is finite state does *not* imply that p has finite behaviour.

i) Give an example that shows: p is finite state does *not* imply that p is deterministic.

j) Give an example that shows: p is deterministic does *not* imply that p has finite behaviour.

k) Give an example that shows: p is deterministic does *not* imply that p is finite state.

1.7 Preorders

a) In the following proof exert every step in detail, and indicate in each step which argument (definition, property, ...) you use.

Prove that the following holds:

$$i \leq_{ior} s \implies i \leq_{iot} s$$

2 Behaviour Expressions

2.1 Parallel composition

Consider the processes P and Q, which are represented as labelled transition systems in Figure 5 with label set $L = \{a, b\}$.

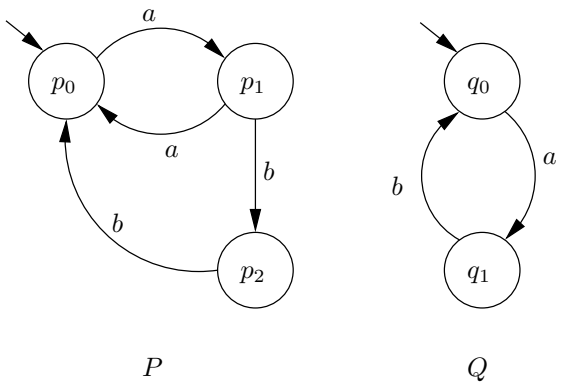


Figure 5:

For the following tasks, show for each state in the composition from which states of the individual systems it is made. So, for example, in the first task the initial state of the composition is (p_0, q_0). From (p_0, q_0), there is a transition labelled a to (p_1, q_1), etc.

a) Draw the LTS for $P \;|[a]|\; Q$.

b) Show that

$$a; b; \mathbf{stop} \;|[b]|\; b; c; \mathbf{stop} \;\equiv\; a; b; c; \mathbf{stop}$$

c) Draw the LTS for $P \;|[b]|\; Q$.

d) Draw the LTS for $(P \;||\; Q) \;|[a]|\; R$, with $R := Q$ (and we have states r_0 and r_1).

2.2 Behaviour expressions

a) Draw the LTS for P, with $P := a; P \;\Box\; b; Q$, and $Q := b; Q$.

b) Draw the LTS for P, with $P := a; b; (\mathbf{hide}\ a\ \mathbf{in}\ P)$.

c) Draw the LTS for P, with $P := a; a; (P \;\Box\; b; P)$.

d) Draw the LTS for P, with $P := c; Q \;||\; R$, $Q := a; Q \;\Box\; b; c; Q$, and $R := b; R \;\Box\; a; b; R \;\Box\; a; a; \mathbf{stop}$.

e) Draw the LTS for P, with $P := Q \;|[a]|\; a; (\mathbf{hide}\ a\ \mathbf{in}\ Q)$, and $Q := c; Q \;\Box\; a; b; Q$.

f) Draw the LTS for P, with $P := (\mathbf{hide}\ a\ \mathbf{in}\ Q) \;||\; R$, $Q := b; (a; Q \;\Box\; b; c; Q)$, and $R := b; R \;\Box\; a; \mathbf{stop}$.

2.3 Behaviour expressions and equivalences

a) Draw the transition system of P,
with $P := a; b; \tau; P \;\Box\; c; (d; \mathbf{stop} \Box e; Q)$ and $Q := c; Q$.

b) Draw the transion system of $(a; \mathbf{stop} \;|||\; b; \mathbf{stop} \;|||\; c; \mathbf{stop}) \;|[a,b]|\; a; b; \mathbf{stop}$

c) Show that $a; b; \mathbf{stop} \;|[b]|\; b; c; \mathbf{stop} \;\equiv\; a; b; c; \mathbf{stop}$

Note: $\equiv$ is *isomorphism*, i.e., the transition systems are the same modulo the names of the states.

d) Show that $\mathbf{hide}\ b, d\ \mathbf{in}\ P \;|[b,d]|\; Q \;\approx_{tr}\; R$
where $P := a; b; d; P$, $Q := b; c; d; Q$, $R := a; c; R$.

2.4 Behaviour expressions and equivalences

a) Draw the transition system of P,
with $P := a; P \;\Box\; a; Q$ and $Q := c; Q$.

b) Show that

$$\mathbf{hide}\ b, c\ \mathbf{in}\ \;(\;(\; a; b; \mathbf{stop} \;|[b]|\; b; c; \mathbf{stop}\;)\;|[c]|\; c; d; \mathbf{stop}\;)\;\approx_{tr}\; a; d; \mathbf{stop}$$

Note: $\approx_{tr}$ is *trace equivalence*: $p \approx_{tr} q$ iff $traces(p) = traces(q)$.

c) Give a behaviour expression without parallellism which is trace equivalent to

$$P \;|[a,b]|\; Q$$

where $P := a; x; y; b; P$, $\;Q := a; u; v; b; Q$.

2.5 Associativity

a) In the following proof exert every step in detail, and indicate in each step which argument (definition, property, ...) you use.

Let P, Q, R be processes described using behaviour expressions with label set L. Given some set $G \subseteq L$, prove:

$$(P \;|[G]|\; Q) \;|[G]|\; R \;\equiv\; P \;|[G]|\; (Q \;|[G]|\; R)$$

b) Let P, Q, R be processes described using behaviour expressions with label set L. Given sets $G_1, G_2 \subseteq L$. Show with a counter example that in general the following does **not** hold:

$$(P \;|[G_1]|\; Q) \;|[G_2]|\; R \;\equiv\; P \;|[G_1]|\; (Q \;|[G_2]|\; R)$$

2.6 Tools for labelled transition systems

Given a behaviour expression $P := Q \;|[a]|\; R \;|[a]|\; R$, with $Q := a; Q'$, $Q' := b; a; Q \;\Box\; b; c; Q'$ and $R := a; R \;\Box\; b; c; R$.

a) Embed this behaviour expression in a LOTOS specification.

b) Generate the transition system with `xeuca` of the CADP tool set, and display your result.

c) Reduce the transition system of the previous question with `xeuca` using *strong equivalence*. Display your result.

d) Find non-determinism in both transition systems of the last two questions using `xeuca`, and give the results. Explain why there is no non-determinism on label a.

e) Compute $P \approx_{tr} R$ using `xeuca`.

3 Testing with ioco

3.1 Testen met ioco

Beschouw de specificatie *CEM* van de espresso-melk machine in figuur 6, met $L_I = \{0.25, 1.00\}$, $L_U = \{milk, espr\}$.

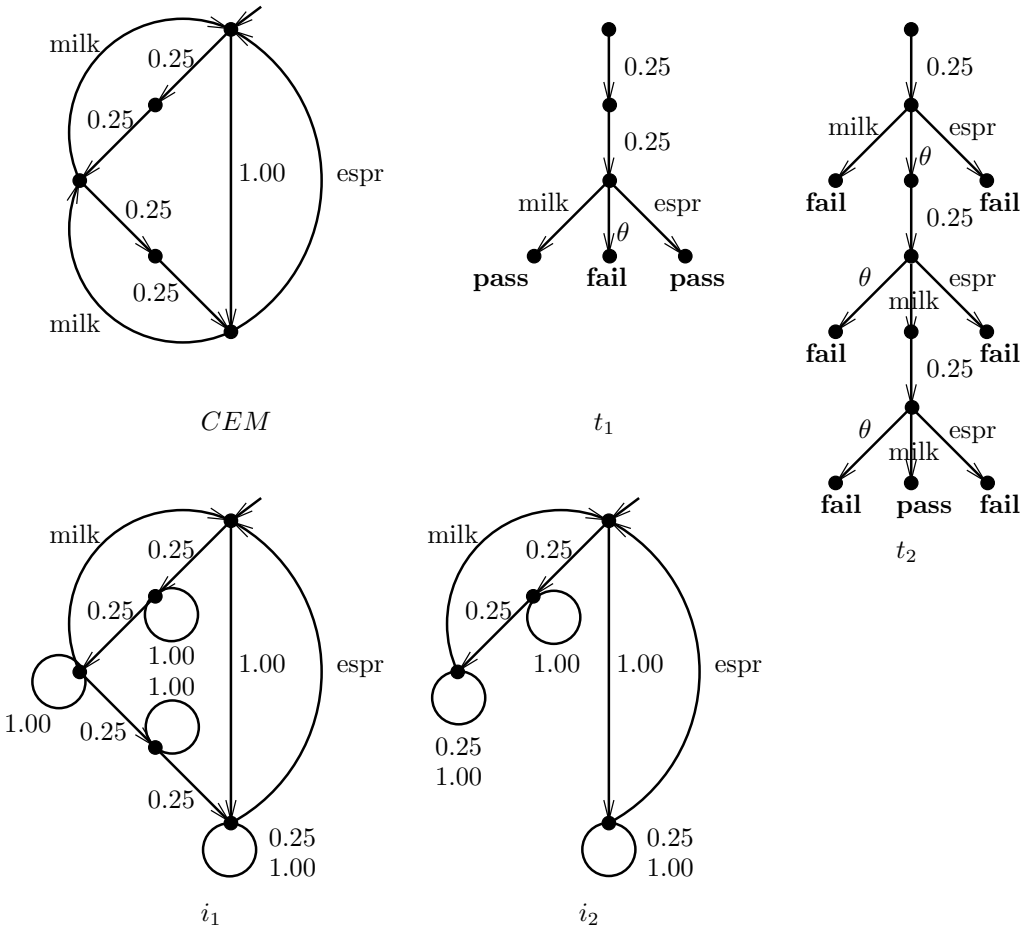


Figure 6:

a) Is $0.25{\cdot}\delta{\cdot}0.25{\cdot}\delta{\cdot}0.25{\cdot}\delta{\cdot}0.25{\cdot}espr$ een *suspension trace* van *CEM*?

b) Is i_1 een **ioco**-correcte implementatie van *CEM*? Zo niet, waarom niet?

c) Is i_2 een **ioco**-correcte implementatie van *CEM*? Zo niet, waarom niet?

d) Geef een **ioco**-correcte implementatie van *CEM* die zowel *milk* als *espr* kan geven.

e) Geef een **ioco**-incorrecte implementatie van *CEM* die zowel *milk* als *espr* kan geven (niet i_1 of i_2).

f) Leid een test case af, volgens het **ioco**-testafleidingsalgoritme, die de **ioco**-incorrecte implementatie van de vorige onderdeel kan detecteren.

g) Leid een test case af, volgens het **ioco**-testafleidingsalgoritme, die controleert of een implementatie niet per ongeluk twee keer *espr* geeft na inworp van één 1.00.

h) Beschrijf in woorden welke fout gedrag de test case t_1 test.

i) Zijn de test cases t_1 en t_2 *sound* voor *CEM* m.b.t. **ioco**? Waarom?

j) Is de test suite $\{t_1, t_2\}$ *sound* voor *CEM* m.b.t. **ioco**? Waarom?

k) Zijn de test cases t_1 en t_2 *exhaustive* voor *CEM* m.b.t. **ioco**? Waarom?

l) Is de test suite $\{t_1, t_2\}$ *exhaustive* voor *CEM* m.b.t. **ioco**? Waarom?

m) Maak LOTOS modellen voor *CEM*, i_1, en i_2, en controleer je resultaten van onderdelen *b*) en *c*) met TORX.

n) Doe hetzelfde voor de gemaakte implementaties in onderdelen *d*) en *e*).

o) Controleer je resultaat van onderdeel *f*) door de gemaakte test case in TORX met de hand uit te voeren, d.w.z. met in mode `manual`.

3.2 iocotesting

Consider the specification EM of an espresso-and-milk machine in Figure 7; $L_I = \{0.25,1.00\}$, $L_U = \{\text{milk,espr}\}$.

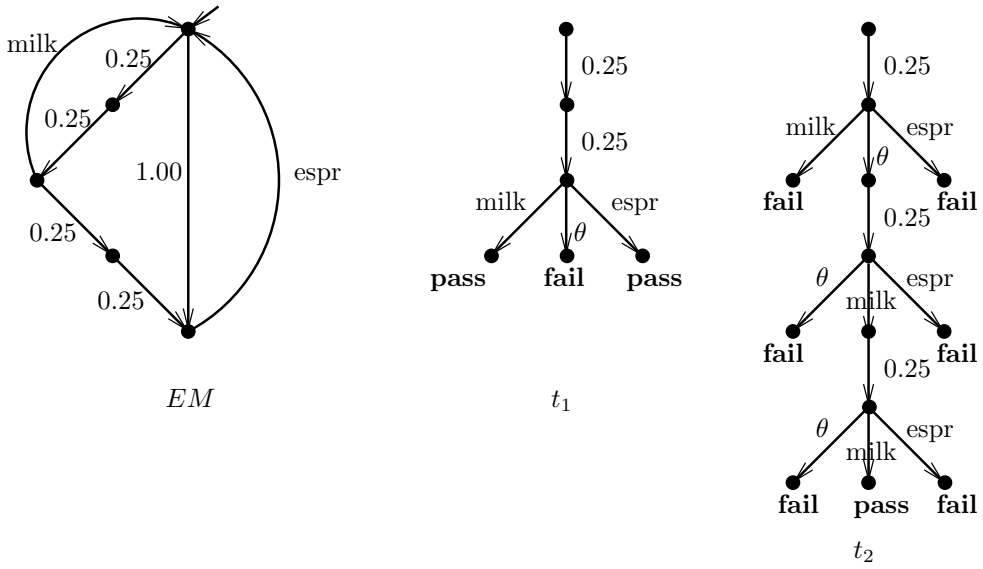


Figure 7:

a) Give a correct and an erroneous espresso-machine implementation, according to the implementation relation **ioco**, as input-output transition systems, and give the arguments why they are correct/erroneous.

b) Derive, step-by-step according to the **ioco** test derivation algorithm, a test case which is able to detect (i.e., gives the verdict **fail** with) the erroneous implementation that you have given in (a).

c) Are the test cases t_1 and t_2 in Figure 7 sound for EM with respect to **ioco**? Why?

d) Is the test suite $\{t_1, t_2\}$ in Figure 7 sound for EM with respect to **ioco**? Why?

e) Is the test suite $\{t_1, t_2\}$ in Figure 7 exhaustive for EM with respect to **ioco**? Why?

3.3 ioco testing

The owner of a Dutch newspaper stand wishes to order a machine to sell newspapers. He is selling the "De Telegraaf" for € 1,= and "NRC Handelsblad" for € 2,=. To be able to ask an offer from different automatic newspaper selling machine manufacturers, he first makes a specification of the system he would like to have.

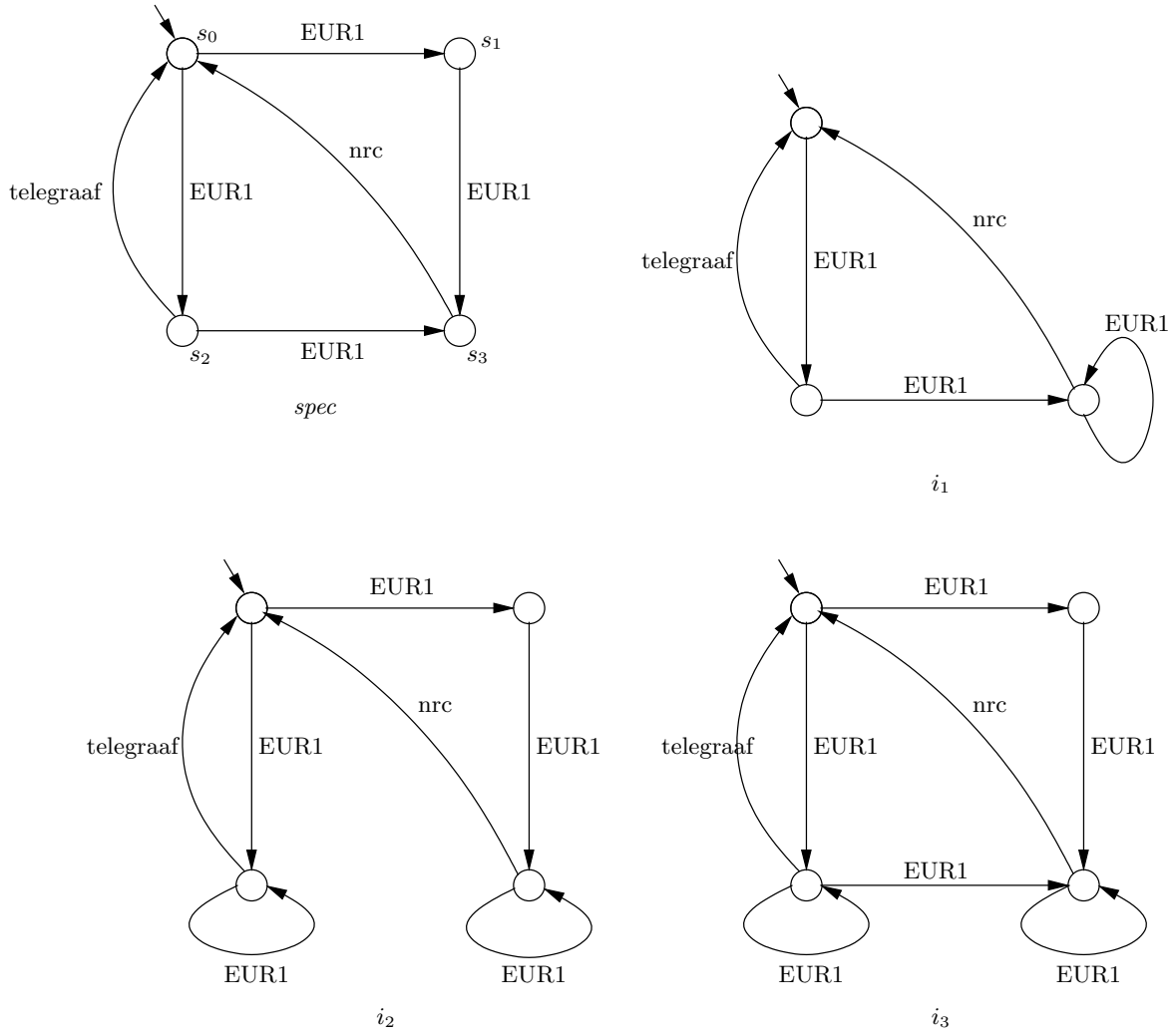


Figure 8:

The specification *spec* is given in Figuur 8. The stand owner explicitly asks the manufacturers to deliver only **ioco**-conforming implementations, with $L_I = \{\text{EUR1}\}$ and $L_U = \{\text{telegraaf}, \text{nrc}\}$:

$$i \textbf{ ioco } s \quad \Leftrightarrow_{\text{def}} \quad \forall \sigma \in \mathit{Straces}(s) : \ \mathit{out}(\, i \textbf{ after } \sigma \,) \subseteq \mathit{out}(\, s \textbf{ after } \sigma \,)$$

Some manufacturers make him an offer. The machines that they offer are given in Figuur 8 as i_1, i_2 and i_3.

a) Which states of *spec* are *quiescent*?

b) Which of the implementations i_1, i_2 or i_3 are **ioco**-correct?

c) Check your answers on the previous questions using TorX. (*spec*, i_1, i_2, and i_3 are given in directory `newspaper` in `.aut`-format.)

Of course, an important requirement is that the machines will never supply newspapers without payment. Moreover, the machines should be able to deliver multiple newpapers after multiple payments.

d) Make a test case using the **ioco**-test generation algorithm which checks whether an implementation can deliver a free newspaper in the initial state.

e) Make a test case using the **ioco**-test generation algorithm which checks whether after inserting ∈ 4,= in total, the machine delivers the NRC twice.

f) Using the **ioco**-test generation algorithm all terminal states of the test case in (*e*) will be labelled either **pass** or **fail**, but perhaps there are terminal states which could be better labelled **inconclusive**. Do you have such states in your test case in (*e*), and, if yes, which states should be labelled **inconclusive**? Why?

The publishers of "De Telegraaf" and of "NRC Handelsblad" also want to make an offer, but these machines turn out to sell only their own newspapers.

g) Make a system which the publisher of "NRC Handelsblad" might deliver, i.e., make an **ioco**-conforming implementations which has NRC as only output.

h) Make a system which the publisher of "De Telegraaf" might deliver, i.e., make an **ioco**-conforming implementations which has De Telegraaf as only output.

i) Check your answers on the previous questions using TorX.

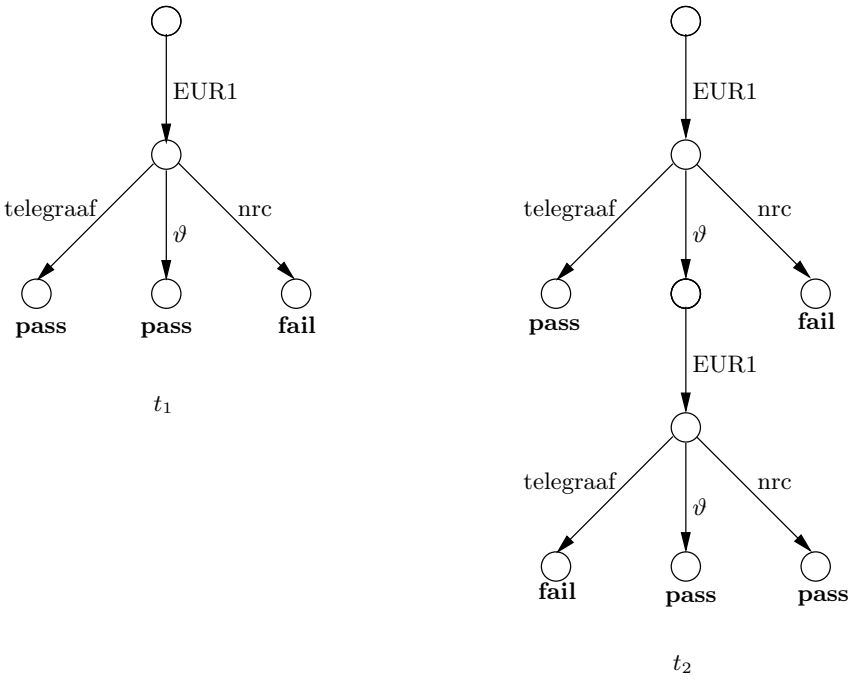


Figure 9:

The manufacturer of machine i_3 offers the machine together with two test cases t_1 en t_2, see Figure 9, to convince our newspaper salesman that his machine is correct.

j) Give all the test runs of applying the tests t_1 and t_2 to i_3. What is the verdict of test execution for these tests? What is the verdict of test execution for the test suite $\{t_1, t_2\}$?

k) Check, as far as possible, your answer in the previous question by using TorX: execute the tests in the `manual`-mode of TorX.

l) Is it possible to derive test cases t_1 and t_2 from *spec* using the **ioco**-test generation algorithm?

m) Are the test cases t_1 and t_2 *sound* with respect to *spec*?

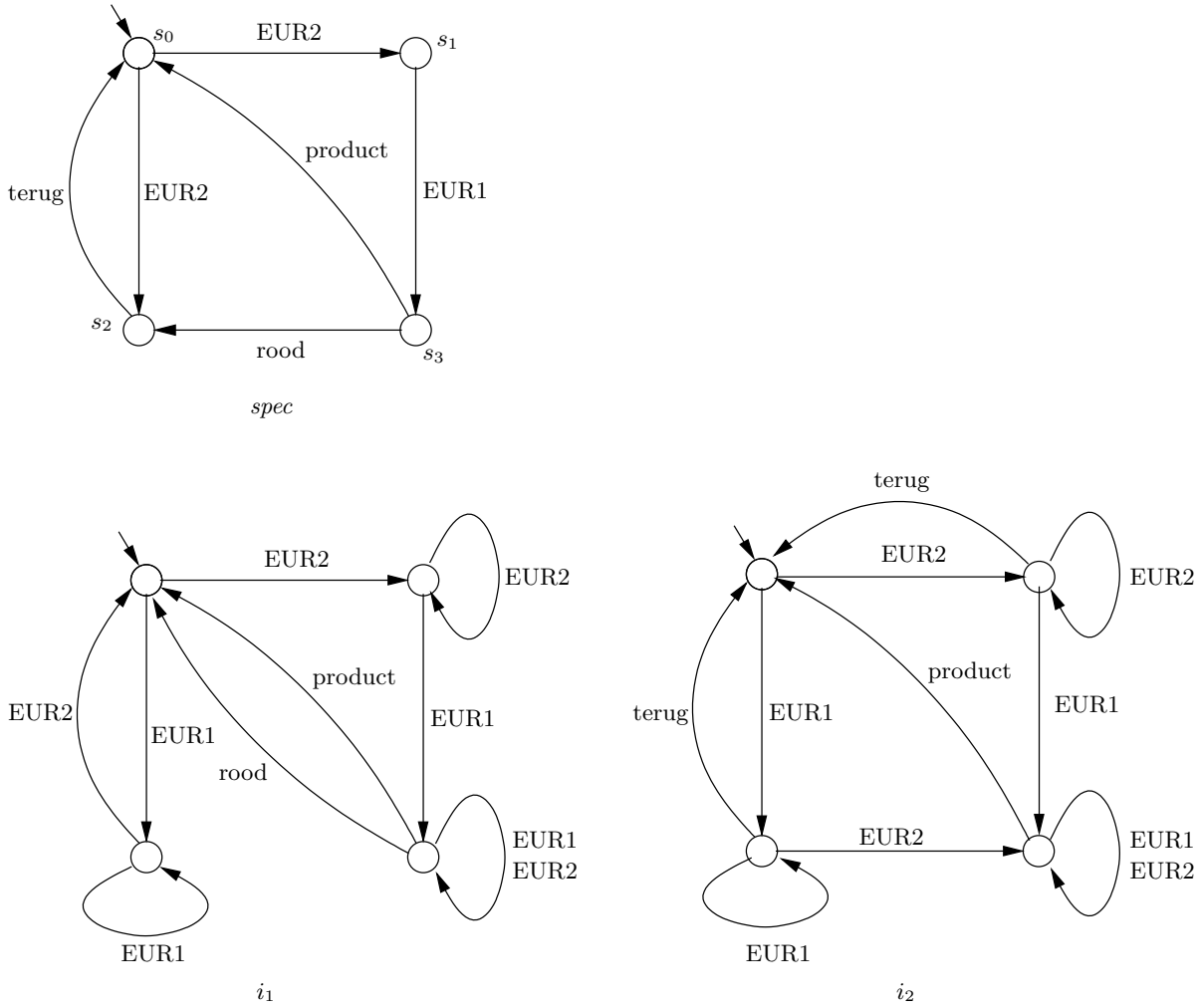
n) Is the test suite $\{t_1, t_2\}$ *sound* with respect to *spec*?

o) Are the test cases t_1 and t_2 *exhaustive* with respect to *spec*?

p) Is the test suite $\{t_1, t_2\}$ *exhaustive* with respect to *spec*?

3.4 Testen met ioco

Beschouw de product-machines die in figuur 10 als gelabeld transitiesysteem gegeven zijn, met $L_I = \{EUR1, EUR2\}$ en $L_U = \{product, rood, terug\}$. De product-machine accepteert € 1 en € 2 munten en geeft dan een product af. De producten kunnen op zijn; dan gaat een rode lamp branden en het geld wordt teruggegeven. In figuur 10 is de specificatie van de product-machine *spec* gegeven, samen met twee implementaties i_1 en i_2.



spec

i_1

i_2

Figure 10:

a) Welke toestanden van *spec* zijn *quiescent*?

b) Welke van de implementaties i_1 en i_2 zijn **ioco**-correct?

c) Natuurlijk mogen producten niet zonder betalen geleverd worden. Maak een test case m.b.v. het **ioco**-testafleidingsalgoritme die test of een implementatie in de begintoestand niet per ongeluk gratis producten levert.

d) Maak een test case m.b.v. het **ioco**-testafleidingsalgoritme die test of een implementatie na invoer van totaal $\leqslant 6$,$= \leqslant$ inderdaad twee keer een product kan leveren.

e) Bedenk een **ioco**-correcte implementatie die nooit een product levert.

f) Bedenk een **ioco**-incorrecte implementatie die een product kan leveren, rood kan worden, en het geld teruggeeft.

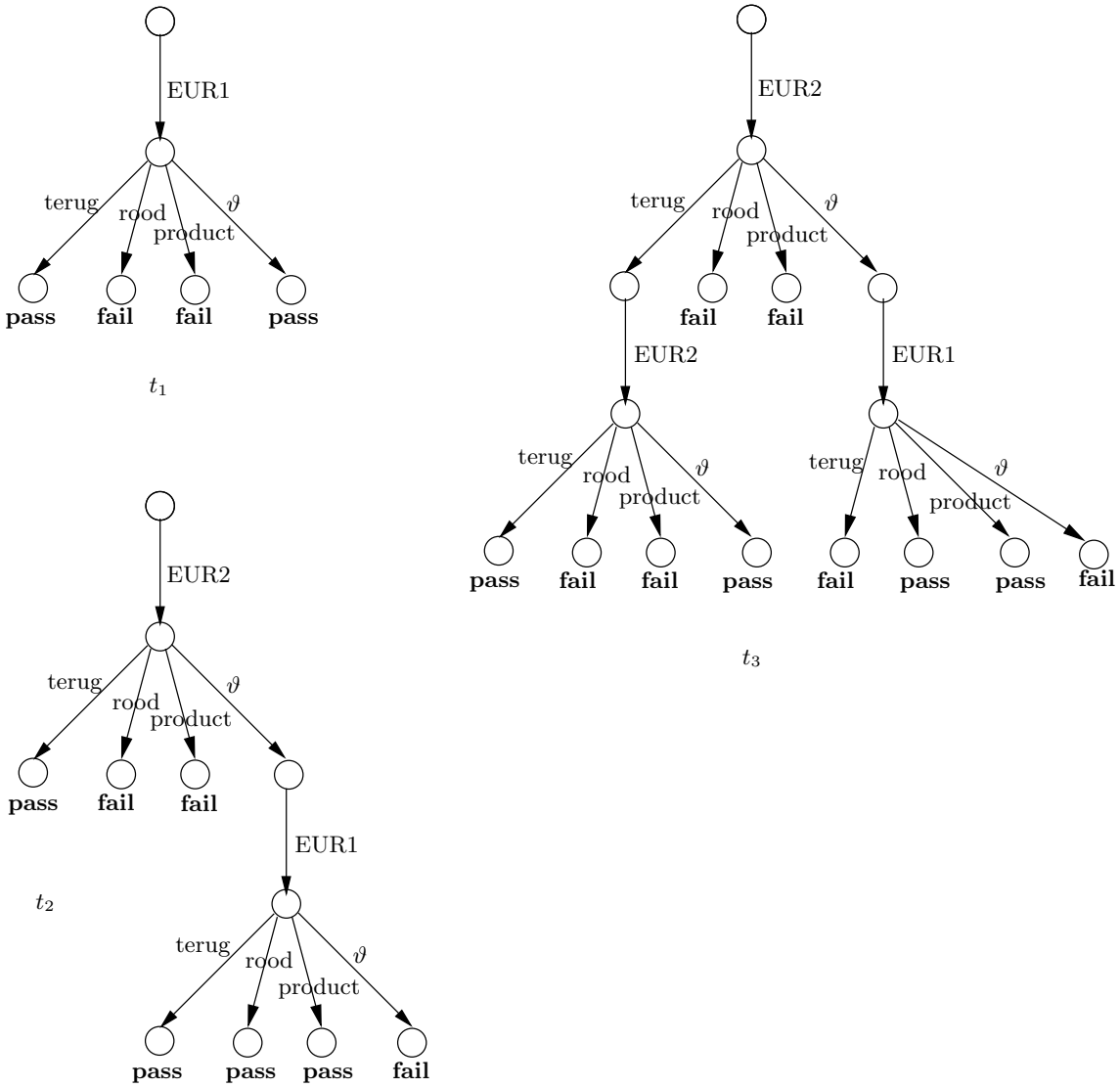


Figure 11:

g) In figuur 11 staan 3 test cases t_1, t_2 en t_3. Geef alle *test runs* van toepassing van t_2 op i_1 en t_3 op i_2. Wat is het resultaat?

h) Kunnen t_1, t_2 en t_3 afgeleid worden met het **ioco**-testafleidingsalgoritme?

i) Zijn t_1, t_2 en t_3 *sound* m.b.t. *spec*?

j) Is de testverzameling $\{t_1, t_2, t_3\}$ *exhaustive* m.b.t. *spec*?

k) Controleer je resultaten van de onderdelen *b*), *e*) en *f*) met behulp van TorX. Maak Lotos modellen van *spec*, i_1 en i_2, en gebruik het `LOTOStemplate`.

3.5 ioco testing

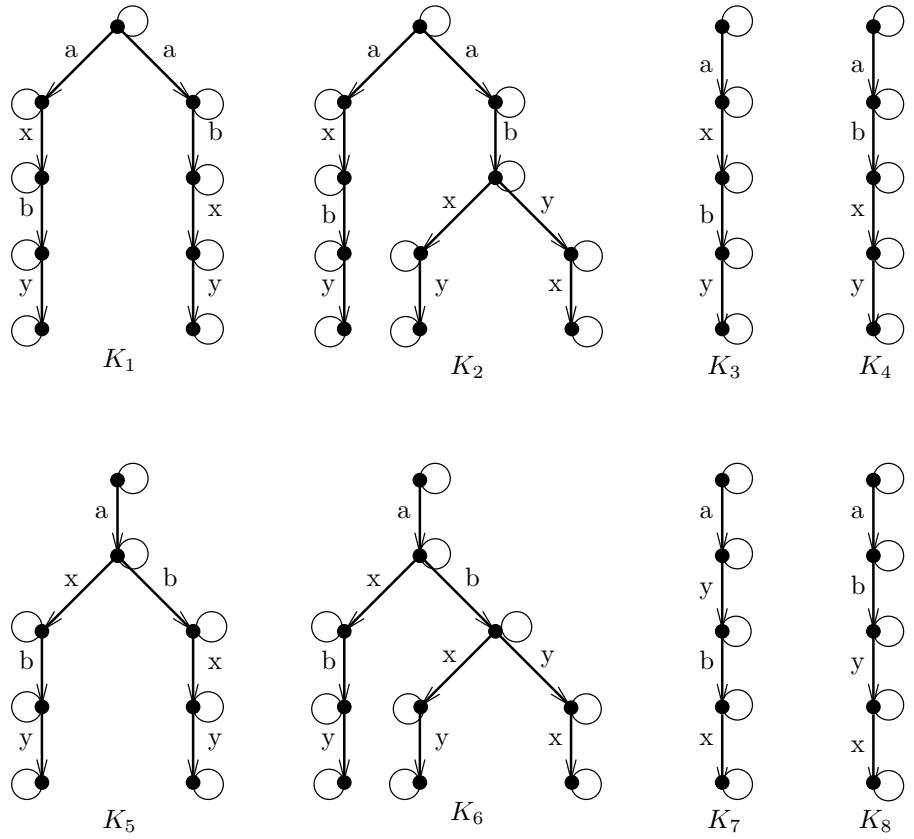


Figure 12:

Considerer the input-output transition systems k_1–K_8 with $L_I = \{a, b\}$ and $L_U = \{x, y\}$ in Figure 12. Which systems are correct **ioco** implementations of each other? Present the results in a matrix. Give an argument for those pairs which are not correct according to **ioco**.

Check your results with TorX. Use the automata in directory `k18`.

3.6 ioco testing: the *coffee*-example

Consider the labelled transition system given in Figure 13 specifying (again) a coffee machine. The label *button* is an input; *coffee*, *espresso* and *cappuccino* are outputs ($L_I = \{button\}$, $L_U = \{coffee, espresso, cappuccino\}$).

The coffee machine is also given in the file `spec1.aut`. In this file the labelled transition system is given in so-called *Aldebaran*-format (extension `.aut`):

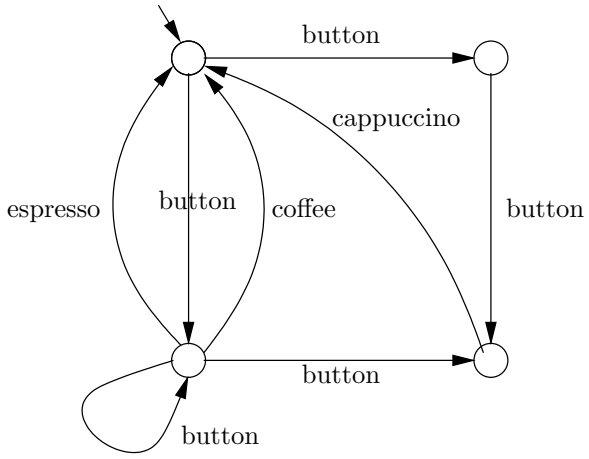


Figure 13: Coffee machine as labelled transition system

```
des (0, 8, 4)
(0, button, 1)
(0, button, 3)
(1, button, 2)
(2, cappuccino, 0)
(3, button, 2)
(3, button, 3)
(3, espresso, 0)
(3, coffee, 0)
```

A labelled transition systems in `.aut`-format is simply a list of transitions with source state - transition label - destination state. The first line of an `.aut`-file specifies (from left to right) the initial state, the number of transitions, and the number of states. It is advisable that the initial state always has number 0.

To view an `.aut`-file you may use the Eucalyptus Toolset by calling `xeuca &`. Click with the left mouse button on the `.aut`-file you would like to see and select `Visualize→Edit`. Subsequently you may have to move the objects a little bit in order to obtain a readable transition system.

In the files for this assignment you will also find five implementations given in `.aut`-format, viz. the files `impl3.aut`–`impl7.aut`. In this exercise we will test these labelled transition system implementations (in `.aut`-format) with respect to the labelled transition system specification `spec1.aut`

a) *Check the correctness manually* Investigate the five implementations closely. Is each of these five given implementations an **ioco**-correct implementation of the coffee machine given in Figure 13? Which one(s) are not **ioco**-correct w.r.t. the specification, and why?

b) *Checking the correctness with* TorX Check the correctness of the given implementations with the test tool TorX. Follow the directions for using TorX. Compare the results with the manual checking.

c) *Make your own correct implementation* Make a new implementation (by making a new `.aut`-file) that is **ioco**-correct with respect to the specification `spec1.aut` but that cannot give *cappuccino*. Test this implementation with `xtorx` with respect to `spec1`. Is it indeed a correct implementation?

d) *Make your own incorrect implementation* Make a new implementation (by making a new `.aut`-file) that is **ioco**-incorrect with respect to the specification `spec1.aut`, but that can produce *coffee* and *espresso*. Test this implementation with `xtorx` with respect to `spec1`. Analyse why it is not

correct and give the (shortest) MSC (message sequence chart) produced by TorX indicating the error.

3.7 ioco testing

Consider a system with three coloured lights (*red*, *yellow*, and *green*) which can flash after a *button* has been pushed. Each light flashes exactly once after the *button* has been pushed, and after all three lights have flashed the *button* can be pushed again to repeat the procedure. The order in which the lights flash is not important.

A specification of this system is given in `light_system.lot` in the directory `light`. The specification is given in the *process algebra* Lotos.

a) Argue whether a system in which the lights always flash in the order *red*, *yellow*, *green*, is an **ioco**-correct implementation of `light_system.lot`.

b) Write a Lotos model for the implementation under (*a*); verify your answer given in (*a*) with TorX.

c) Argue whether a system in which, after *button*, each light flashes once or zero times, is an **ioco**-correct implementation of `light_system.lot`.

d) Write a Lotos model for the implementation under (*c*); verify your answer given in (*c*) with TorX.

e) Argue whether a system in which the button can be pushed again immediately after the first flash, aborting the series of three flashes and starting a new series of three flashes, is an **ioco**-correct implementation of `light_system.lot`.

f) Test the implementation model `light_impl1.lot` with respect to the specification `light_system.lot`; do you think it **ioco**-correct or erroneous? If erroneous, what is wrong?

3.8 ioco testing

In Figure 14 you see the specification *spec* of yet another coffee machine. It can produce either espresso or cappuccino. Action *?coin* represents insering a coin, *?esp* represents pressing the espresso-button, and *?capp* represents pressing the cappuccino-button. Action *!coin* represents returning the coin, *!esp* represents obtaining espresso, and *!capp* represents obtaining cappuccino. You also find three implementations (*i1* to *i3*) given as IOTSs.

a) Which states of *spec* are quiescent?

b) Which of the transition systems in Figure 14 are deterministic?

c) Which of the implementations *i1*, *i2* or *i3* are **ioco**-correct? Give a suspension trace for every non-**ioco**-correct implementation that shows the non-conformance.

d) Make an **ioco**-correct implementation which can only produce cappuccino.

e) Make a test case t_1 using the **ioco**-test generation algorithm which checks whether an implementation can deliver a cappuccino without paying for it, but with pressing the appropriate buttons.

f) Make a test case t_2 using the **ioco**-test generation algorithm which checks whether an implementation can produce espresso and cappuccino after eachother, and after having inserted the appropriate coins and having pressed the appropriate buttons.

g) Give all test runs of applying the test case t_1 to *i3*. What is the verdict of test execution?

h) Change one of the test cases t_1 or t_2 in such a way that it can *not* be derived from the **ioco**-test generation algorithm, but remains sound.

i) Argue whether there can be a single sound and complete test case for this coffee machine.

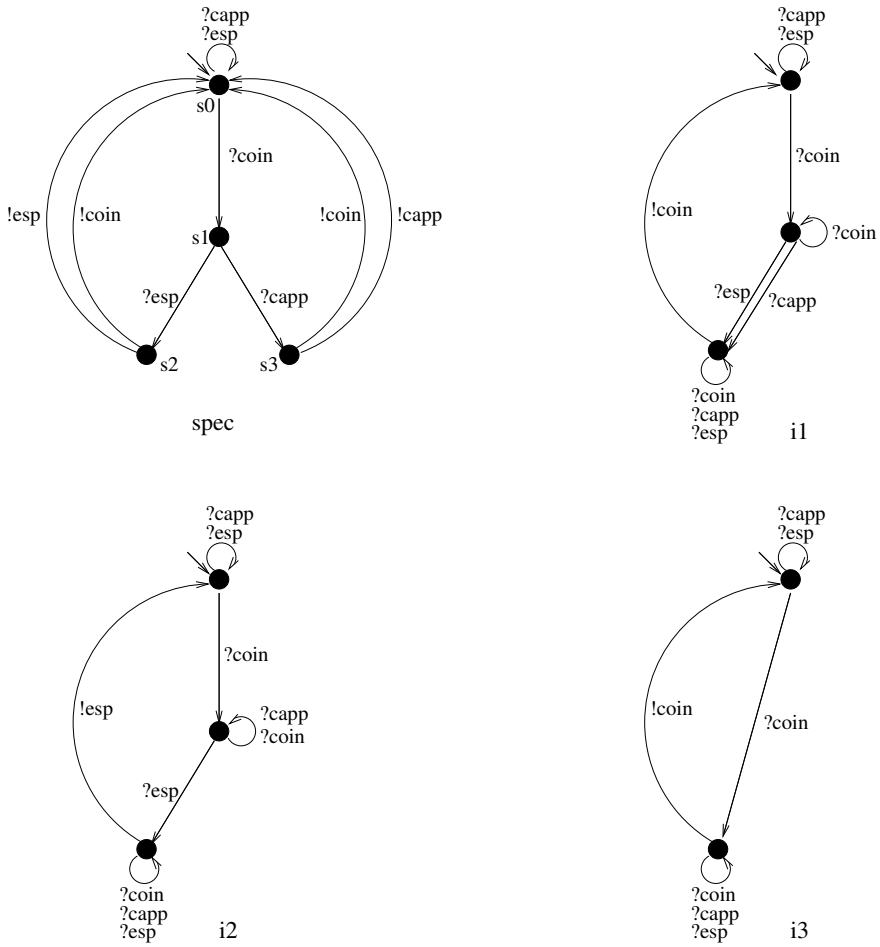


Figure 14: The specification and implementations of the coffee machine.

3.9 Testing with ioco

Consider the specification $s \in \mathcal{LTS}(L_I, L_U)$, and the implementations $i_1, i_2 \in \mathcal{IOTS}(L_I, L_U)$ in Figure 15, where $L = L_I \cup L_U$, with $L_I = \{?b, ?c\}$ and $L_U = \{!espr, !milk, !sugar\}$. The system s (again) specifies a coffee machine, this time producing espresso and cappuccino. If the button $?b$ is pushed an *espresso* is obtained; if the button $?c$ is pushed the machine may produce three ingredients for a cappuccino: *milk*, *sugar*, and *espresso* (chocolate powder must be added manually). In case the machine runs out of *milk* or *sugar*, it may produce just *espresso* when $?c$ is pushed. Cappuccino without *sugar* can be obtained by pushing $?b$ "sufficiently fast" after the *milk* has been produced, however, since labelled transition systems cannot specify real-time properties, the specification of s in Figure 15 abstracts from timing issues like "sufficiently fast".

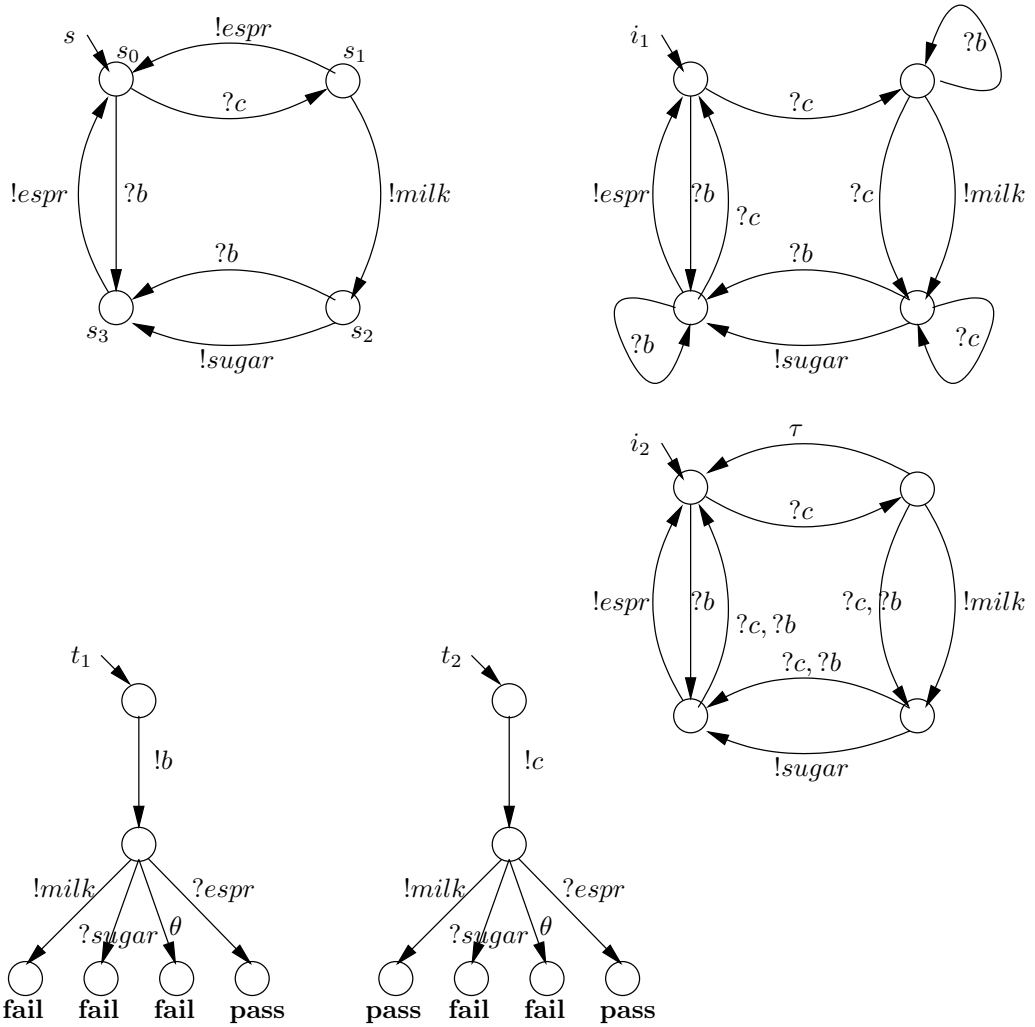


Figure 15:

a) Is s *deterministic*? Consider this question both from the formal point of view (i.e., the definition of determinism), and from the intuitive point of view.

b) Which states of s are *quiescent*?

c) Consider **ioco** as implementation relation:

$$i \textbf{ ioco } s \quad \Leftrightarrow_{\text{def}} \quad \forall \sigma \in Straces(s) : \ out(\, i \textbf{ after } \sigma \,) \subseteq out(\, s \textbf{ after } \sigma \,) \tag{1}$$

Which of the implementations i_1, i_2 are **ioco**-correct with respect to s?

d) Use the *input-output refusal relation* (also called *repetitive quiescent trace preorder*) $\leq_{ior}$ as an implementation relation:

$$i \leq_{ior} s \quad \Leftrightarrow_{\text{def}} \quad \forall \sigma \in (L \cup \{\delta\})^* : \ out(\, i \,\textbf{after}\, \sigma \,) \subseteq out(\, s \,\textbf{after}\, \sigma \,) \tag{2}$$

Which of the implementations i_1, i_2 are $\leq_{ior}$-correct with respect to s?

e) *Prove* that any $\leq_{ior}$-correct implementation is also **ioco**-correct, i.e., prove that $\leq_{ior} \subseteq$ **ioco**.

f) Figure 15 also gives two test cases t_1 and t_2. Give the test runs and determine the verdict of executing the test suite $\{t_1, t_2\}$ on i_2.

g) *Prove* that the test suite $\{t_1, t_2\}$ is not exhaustive for s with respect to **ioco**.
By definition:

T is exhaustive $\quad \Leftrightarrow_{\text{def}} \quad \forall i \in \mathcal{IOTS}(L_I, L_U) : \ i \ \textbf{passes} \ T \ \Rightarrow \ i \ \textbf{ioco} \ s.$

h) Consider the alternative implementation i_3, which uses two separate and independent components for producing *espresso* and *cappuccino*:

$$\begin{array}{lcl} i_3 & := & E \ ||| \ C \\ E & := & ?b; \ E' \\ E' & := & !espr; \ E \ [] \ ?b; \ E' \\ C & := & !milk; \ C' \\ C' & := & !sugar; E' \ [] \ ?b; \ E' \ [] \ ?c; \ C' \end{array}$$

Draw the labelled transition system of i_3.

i) Show that i_3 is not an **ioco**-correct implementation of s.

j) Make a test case using the **ioco**-test generation algorithm which detects i_3 as a non-**ioco**-correct implementation of s. Also express in words which property of s is violated by i_3 and which is detected by your test case.

3.10 A new relation

Let us define a new relation $\approx_{ioco}$, which is defined by:

$$p \approx_{ioco} q \quad \Longleftrightarrow \quad p \ \textbf{ioco} \ q \ \wedge \ q \ \textbf{ioco} \ p$$

a) On which domain(s) is $\approx_{ioco}$ well-defined?

b) Is $\approx_{ioco}$ an equivalence relation? Explain why, or why not.

Hint: Have a look in Section *4.2 Some Variations and Properties of ioco* of the paper *Model Based Testing with Labelled Transition Systems*, which you can find among the Course Documents on BlackBoard, labelled as *alternative reading*.

c) In the following proof exert every step in detail, and indicate in each step which argument (definition, property, ...) you use.

Prove that the following holds:

$$p \approx_{ioco} q \Longrightarrow p \approx_{tr} q$$

d) Show with a counter-example that the following **does not** hold:

$$p \approx_{ioco} q \Longleftarrow p \approx_{tr} q$$

3.11 The uioco-relation

For a definition of **uioco**: see Section *4.2 Some Variations and Properties of ioco* of the paper *Model Based Testing with Labelled Transition Systems*.

a) Prove that the following holds:

$$i \textbf{ uioco } s \Longleftarrow i \textbf{ ioco } s$$

b) Show with a counter-example that the following **does not** hold:

$$i \textbf{ uioco } s \Longrightarrow i \textbf{ ioco } s$$