Testing Techniques 2020 – 2021
Tentamen

January 12, 2021 – 8:30–11:30/12:00 h. – LIN 2 / HG00.058

1 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 Fig. 1, where $L = L_I \cup L_U$, with $L_I = \{?b\}$ and $L_U = \{!espr, !sugar\}$. The system s specifies a coffee machine (what else ...), this time producing different varieties of espresso. If the button $?b$ is pushed three times suffiently fast after each other, the machine just produces an *espresso*. After two times pushing the button an *espresso* with *sugar* is produced, and after having pushed $?b$ once a double *espresso* with *sugar* is produced. For a double *espresso* without *sugar* you have to push the button $?b$ again immediately after the first *espresso* has been produced, in order to "skip" the *sugar* transition.

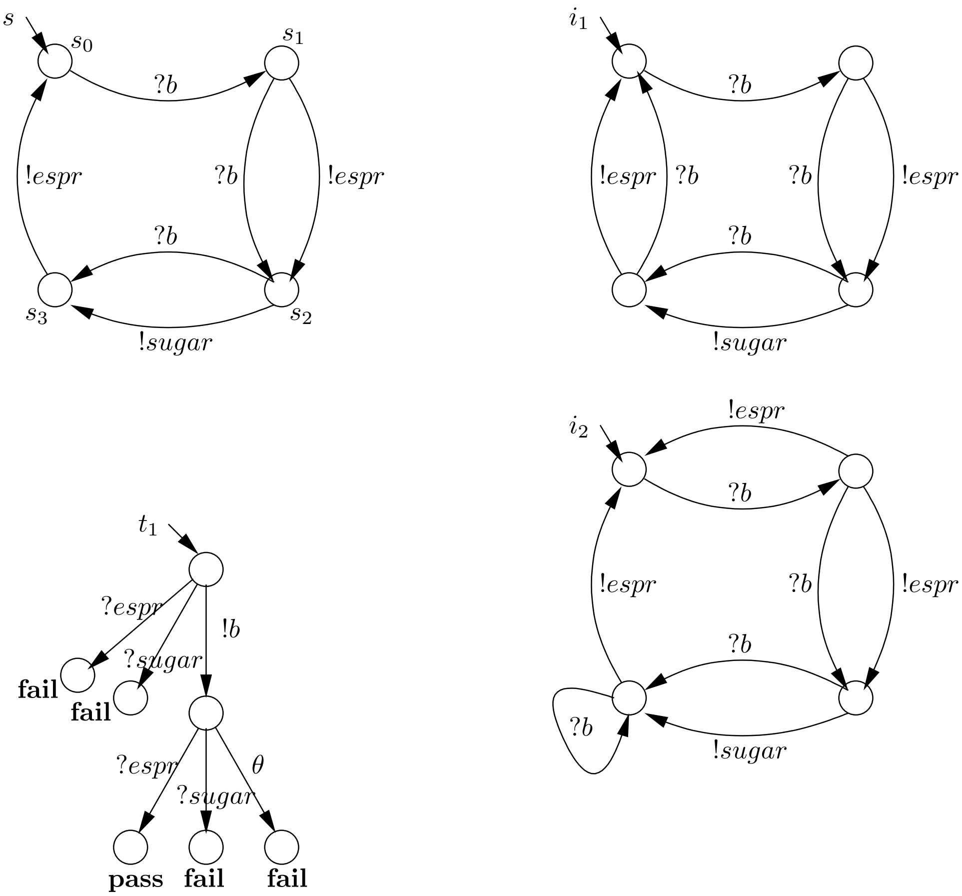

Figure 1:

a. Which states of s are *quiescent*?

Answer
A quiescent state does not have any outgoing transitions labelled with an output or with a τ-transition. Only the initial state s_0 has this property. $\square$

b. Consider **ioco** as implementation relation:

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

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

Answer
i_1 **ioco** s: Add state names $i_{1_0}, i_{1_1}, i_{1_2}, i_{1_3}$ to i_1 analogous to s_0, s_1, s_2, s_3. Then $\mathit{out}(i_{1_k}) \subseteq \mathit{out}(s_k)$ for all states s_k, i_{1_k}, $0 \leq k \leq 3$. Moreover, for all traces σ feasible in both s and i_1, $s \stackrel{\sigma}{\Longrightarrow} s_k$ iff $i_1 \stackrel{\sigma}{\Longrightarrow} i_{1_k}$. It follows that $\forall \sigma \in \mathit{Straces}(s) : \mathit{out}(\, i \textbf{ after } \sigma\,) \subseteq \mathit{out}(\, s \textbf{ after } \sigma\,)$, so i_1 **ioco** s.

i_2 **io$\not$co** s: Consider the trace $?b{\cdot}!\mathit{espr} \in \mathit{Straces}(s)$.
Then $\mathit{out}(\, i_2 \textbf{ after } ?b{\cdot}!\mathit{espr}\,) = \{!\mathit{sugar}, \delta\} \not\subseteq \{!\mathit{sugar}\} = \mathit{out}(\, s \textbf{ after } ?b{\cdot}!\mathit{espr}\,)$. $\square$

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

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

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

Answer
$i_1 \not\leq_{ior} s$: Consider the trace $?b{\cdot}?b{\cdot}?b{\cdot}?b \in (L \cup \{\delta\})^*$. Then $\mathit{out}(\, i_1 \textbf{ after } ?b{\cdot}?b{\cdot}?b{\cdot}?b\,) = \{\delta\} \not\subseteq \emptyset = \mathit{out}(\, s \textbf{ after } ?b{\cdot}?b{\cdot}?b{\cdot}?b\,)$.

$i_2 \not\leq_{ior} s$: We have that $\leq_{ior} \subseteq$ **ioco** (proposition 1.3 in "Model Based Testing with Labelled Transition Systems"; or see next question). Consequently, i_2 **io$\not$co** s implies $i_2 \not\leq_{ior} s$. $\square$

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

Answer
Take arbitrary $i \in \mathcal{IOTS}(L_I, L_U)$ and $s \in \mathit{LTS}(L_I, L_U)$ with $i \leq_{ior} s$. Then we have the following:

	$i \leq_{ior} s$
implies	$(*$ definition of $\leq_{ior}$ $*)$
	$\forall \sigma \in (L \cup \{\delta\})^*: \ \mathit{out}(\, i \textbf{ after } \sigma\,) \subseteq \mathit{out}(\, s \textbf{ after } \sigma\,)$
implies	$(*$ use that $\mathit{Straces}(s) \subseteq (L \cup \{\delta\})^*$ $*)$
	$\forall \sigma \in \mathit{Straces}(s): \ \mathit{out}(\, i \textbf{ after } \sigma\,) \subseteq \mathit{out}(\, s \textbf{ after } \sigma\,)$
implies	$(*$ definition of **ioco** $*)$
	i **ioco** s

$\square$

e. Figure 1 also gives a test case t_1. Give the test runs and determine the verdict of executing t_1 on i_1 and i_2.

Answer
$t_1]|\, i_1 \quad \stackrel{b \cdot espr}{\Longrightarrow} \quad$ **pass**$]|\, i_{1_2}$
So, i_1 **passes** $\{t_1\}$.

$t_1]|\, i_2 \quad \stackrel{b \cdot espr}{\Longrightarrow} \quad$ **pass**$]|\, i_{2_0}$
$t_1]|\, i_2 \quad \stackrel{b \cdot espr}{\Longrightarrow} \quad$ **pass**$]|\, i_{2_2}$
So, i_2 **passes** $\{t_1\}$.

$\square$

f. A test suite T is *exhaustive* for a specification model m iff all **ioco**-incorrect implementations of m are detected by T, i.e.,

$$T \text{ is } \textit{exhaustive} \text{ for } m \quad \Longleftrightarrow_{\text{def}} \quad \forall i \in \mathcal{IOTS}(L_I, L_U):\ i \mathrel{\mathbf{io}\!\!\not{\mathbf{c}}\mathbf{o}} m \ \Rightarrow\ i \textbf{ fails } T$$

Show that the test suite $\{t_1\}$ is *not exhaustive* for s of Fig. 1.
(*Hint: Combine some results of the previous questions.*)

Answer
From the definition of exhaustiveness:

$$\{t_1\} \text{ is not exhaustive iff } \exists i \in \mathcal{IOTS}(L_I, L_U):\ i \mathrel{\mathbf{io}\!\!\not{\mathbf{c}}\mathbf{o}} s \text{ and } i \textbf{ passes } \{t_1\}$$

Above it was shown for i_2 of Fig. 1 that i_2 is such an implementation:
$i_2 \mathrel{\mathbf{io}\!\!\not{\mathbf{c}}\mathbf{o}} s$ and i_2 **passes** $\{t_1\}$. Consequently, $\{t_1\}$ is not exhaustive for s. $\square$

g. Make a test case derived from s using the **ioco**-test generation algorithm that checks whether a double *espresso* without *sugar* can be obtained,

Answer

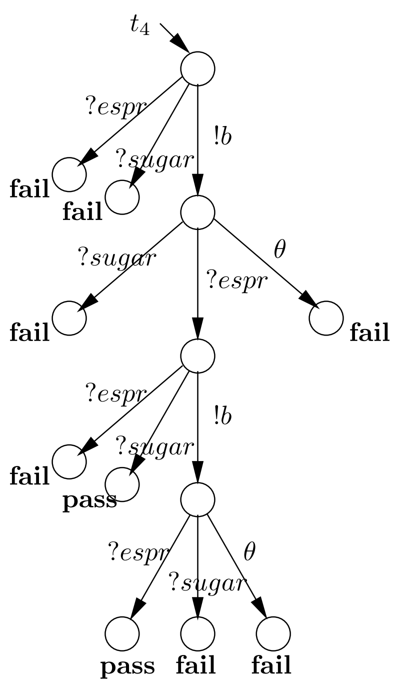

Figure 2:

See test case t_4 in Fig. 2. $\square$

h. Obtaining a double *espresso* without *sugar* may lead to a *race condition* between a user (or tester) trying to push button $?b$ and the machine trying to produce $!sugar$. Explain how the **ioco**-test of the previous question deals with this race condition.

Answer
It was said that "for a double *espresso* without *sugar* you have to push the button $?b$ again immediately after the first *espresso* has been produced, in order to "skip" the *sugar* transition". In the LTS model we cannot specify what 'immediately' means, that is, how fast the user shall be in order to be sure to provide input $?b$ before the machine produces $!sugar$. An LTS model abstracts from timing properties like "suffiently fast" and "immediately". Consequently, in state s_2 both options are possible in the model, the user trying to push button $?b$ and the machine trying to produce $!sugar$.

This also transfers to the test case, i.e., both options are taken into account in the test case: the tester tries to push the button $?b$ but also caters for the possibility that the machine produces output $!sugar$. This leads to possibly nondeterministic test runs, so that, if $!sugar$ occurs during the test run, the test must be repeated, when the goal is to check whether a double *espresso* without *sugar* can be obtained.

□