Testing Techniques 2023 – 2024
Tentamen

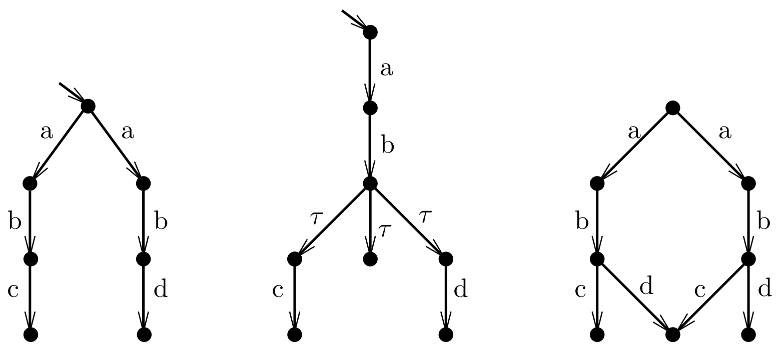
January 22, 2024

- This examination consists of 4 assignments, with weights 2, 2, 3, and 4, respectively.
- The exam has 6 pages, numbered from 1 to 6.
- You are not allowed to use any material during the examination, except for pen and paper, and
  - the paper: *Tretmans: Model Based Testing with Labelled Transition Systems* (38 pages);
  - the slide set: *Vaandrager: Black Box Testing of Finite State Machines* (152 slides);
  - the slide set: *Kruger, Vaandrager: Model Learning* (115 slides).
- Use one or more separate pieces of paper per assignment.
- Write clearly and legibly.
- Give explanations for your answers to open questions, but keep them concise.
- We wish you a lot of success!

Grading: Total

assignment	1	2	3	4	grade
points	max 20	max 20	max 30	max 40	total/11

1 Equivalence



P_1 P_2 P_3

Figure 1:

Consider the processes P_1, P_2, and P_3, which are represented as labelled transition systems in Figure 1, with labelset $L = \{a, b, c, d\}$.

Consider the following definitions:

$$
\begin{array}{rcl}
p \leq_{tr} q & \Longleftrightarrow_{\text{def}} & \mathit{traces}(p) \subseteq \mathit{traces}(q) \\
p \approx_{tr} q & \Longleftrightarrow_{\text{def}} & p \leq_{tr} q \text{ and } q \leq_{tr} p \\
p \approx_{ct} q & \Longleftrightarrow_{\text{def}} & \mathit{traces}(p) = \mathit{traces}(q) \text{ and } \mathit{Ctraces}(p) = \mathit{Ctraces}(q) \\
p \approx_{te} q & \Longleftrightarrow_{\text{def}} & \forall \sigma \in L^*,\ \forall A \subseteq L : \\
 & & \quad p \textbf{ after } \sigma \textbf{ refuses } A \text{ iff } q \textbf{ after } \sigma \textbf{ refuses } A \\
p \textbf{ after } \sigma \textbf{ refuses } A & \Longleftrightarrow_{\text{def}} & \exists p' : p \overset{\sigma}{\Longrightarrow} p' \text{ and } \forall a \in A \cup \{\tau\} : p' \overset{a}{\not\longrightarrow} \\
\mathit{Ctraces}(p) & =_{\text{def}} & \{\ \sigma \in L^* \mid p \textbf{ after } \sigma \textbf{ refuses } L\ \}
\end{array}
$$

Compare the processes P_1, P_2, and P_3 according to

a. trace equivalence $\approx_{tr}$;

Answer
$P_1 \approx_{tr} P_2 \approx_{tr} P_3$,
because $\mathit{traces}(P_1) = \mathit{traces}(P_2) = \mathit{traces}(P_3) = \{\epsilon, a, a{\cdot}b, a{\cdot}b{\cdot}c, a{\cdot}b{\cdot}d\}$. □

b. completed trace equivalence $\approx_{ct}$;

Answer
$\mathit{Ctraces}(P_1) = \mathit{Ctraces}(P_3) = \{a{\cdot}b{\cdot}c, a{\cdot}b{\cdot}d\}$,
but $\mathit{Ctraces}(P_2)$ also contains $a{\cdot}b$: P_2 **after** $a{\cdot}b$ **refuses** L .
On the other hand, not ($P_{1,3}$ **after** $a{\cdot}b$ **refuses** L),
thus, $P_1 \approx_{ct} P_3$, $P_1 \not\approx_{ct} P_2$, and $P_2 \not\approx_{ct} P_3$. □

c. testing equivalence $\approx_{te}$.

Answer
$P_1 \not\approx_{ct} P_2$ and $P_2 \not\approx_{ct} P_3$, so also $P_1 \not\approx_{te} P_2$ and $P_2 \not\approx_{te} P_3$:
P_2 **after** $a{\cdot}b$ **refuses** L but not ($P_{1,3}$ **after** $a{\cdot}b$ **refuses** L).

Moreover, $P_1 \not\approx_{te} P_3$, because P_1 **after** $a{\cdot}b$ **refuses** $\{c\}$,
whereas not (P_3 **after** $a{\cdot}b$ **refuses** $\{c\}$).

□

d. Prove that $\approx_{ct}$ is strictly stronger than $\approx_{tr}$, i.e., $\approx_{ct} \subset \approx_{tr}$ and $\approx_{ct} \neq \approx_{tr}$.

Answer
$\approx_{ct}$ is stronger than $\approx_{tr}$ as follows directly from their definitions:

$$\begin{array}{lcl} p \approx_{ct} q & \Longleftrightarrow_{\text{def}} & \mathit{traces}(p) = \mathit{traces}(q) \text{ and } \mathit{Ctraces}(p) = \mathit{Ctraces}(q) \\ & \text{implies} & \mathit{traces}(p) = \mathit{traces}(q) \quad \Longleftrightarrow_{\text{def}} \quad p \approx_{tr} q \end{array}$$

Strictness follows from P_1 and P_2: from *a.* we have $P_1 \approx_{tr} P_2$, and from *b.* we have $P_1 \not\approx_{ct} P_2$.

□

Grading: Assignment 1

a	b	c	d	points
5	5	5	5	max 20

2 Conformance

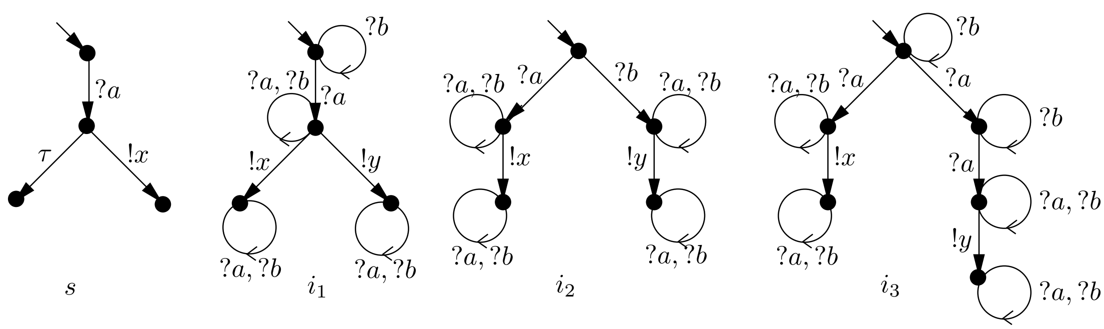


Figure 2:

Consider the labelled transition systems in Fig. 2 with $L_I = \{?a.?b\}$ and $L_U = \{!x, !y\}$.

a. Which of the implementations i_1, i_2, i_3 are **uioco**-conforming to specification s, and why?

Answer
$i_1\ \not\mathbf{uioco}\ s$: $\mathit{out}(\, i_1 \textbf{ after } a\,) = \{!x, !y\} \not\subseteq \mathit{out}(\, s \textbf{ after } a\,) = \{!x, \delta\}$.

i_2 **uioco** s: $\forall \sigma \in \mathit{Utraces}(s) : \ \mathit{out}(\, i_2 \textbf{ after } \sigma\,) \subseteq \mathit{out}(\, s \textbf{ after } \sigma\,)$,
in particular, $\mathit{out}(\, i_2 \textbf{ after } a\,) = \{!x\} \subseteq \mathit{out}(\, s \textbf{ after } a\,) = \{!x, \delta\}$, and $b \notin \mathit{Utraces}(s)$.

i_3 **uioco** s: $\forall \sigma \in \mathit{Utraces}(s) : \ \mathit{out}(\, i_3 \textbf{ after } \sigma\,) \subseteq \mathit{out}(\, s \textbf{ after } \sigma\,)$,
in particular, $\mathit{out}(\, i_3 \textbf{ after } a\,) = \{!x, \delta\} \subseteq \mathit{out}(\, s \textbf{ after } a\,) = \{!x, \delta\}$, and $a{\cdot}a \notin \mathit{Utraces}(s)$. □

b. For each of the implementations i_j, $j = 1, 2, 3$, give a test case t_j, if such a test case exists, such that

- t_j is generated from s using the **uioco**-test generation algorithm; and
- t_j fails with i_j.

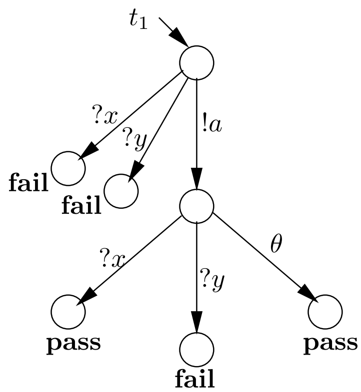


Figure 3: Test case for i_1.

Answer

Since i_2 **uioco** s and i_3 **uioco** s, all tests that are generated from s using the **uioco**-test generation algorithm, will pass, so a test case that fails with i_2 or i_3 does not exist.

For i_1, a test case that provides $!a$ and then observes either $?x$, leading to **pass**, or $?y$, leading to **fail**, or θ, leading to **pass**, can be generated from s, see Fig. 3, and moreover, it fails with i_1:

$t_1 \rceil\!| \, i_1 \xRightarrow{a \cdot x} \textbf{pass} \rceil\!| \, i_{1_3}, \;\; t_1 \rceil\!| \, i_1 \xRightarrow{a \cdot y} \textbf{fail} \rceil\!| \, i_{1_4}, \;$ so $\; i_1$ **fails** t_1.

□

Grading: Assignment 2

a	b	points
12	8	max 20

3 Model-Based Testing

Consider the labelled transition systems s, i_1, and i_2 in Fig. 4, that represent printers. A user can submit a printer job with $?job$, after which the printer indicates whether the submitted job is well-formed or not, via $!ack$ and $!rej$, respectively. A well-formed job can be printed using the $?print$-command, which produces the $!printed$ output. During the process, a user can $?abort$ the printing, after which the printer will go to the initial state again, except if after the $?print$-command the user is too slow with her $?abort$, and the printing has already started and cannot be aborted anymore.

a. Which states of s are *quiescent*, and why?

Answer
The states s_0 and s_2 are quiescent, they do not have any output or internal-transition: $\forall x \in L_U \cup \{\tau\} : s_i \overset{x}{\not\longrightarrow}$, with $i = 0$ or $i = 2$.

□

b. Is s *deterministic*, and why?

Answer
s is non-deterministic: state s_3 has two transitions labelled $?abort$, which makes that s **after** $?job{\cdot}!ack{\cdot}?print{\cdot}?abort = \{s_0, s_2\}$, with $|\{s_0, s_2\}| \not\leq 1$, so s is not deterministic.

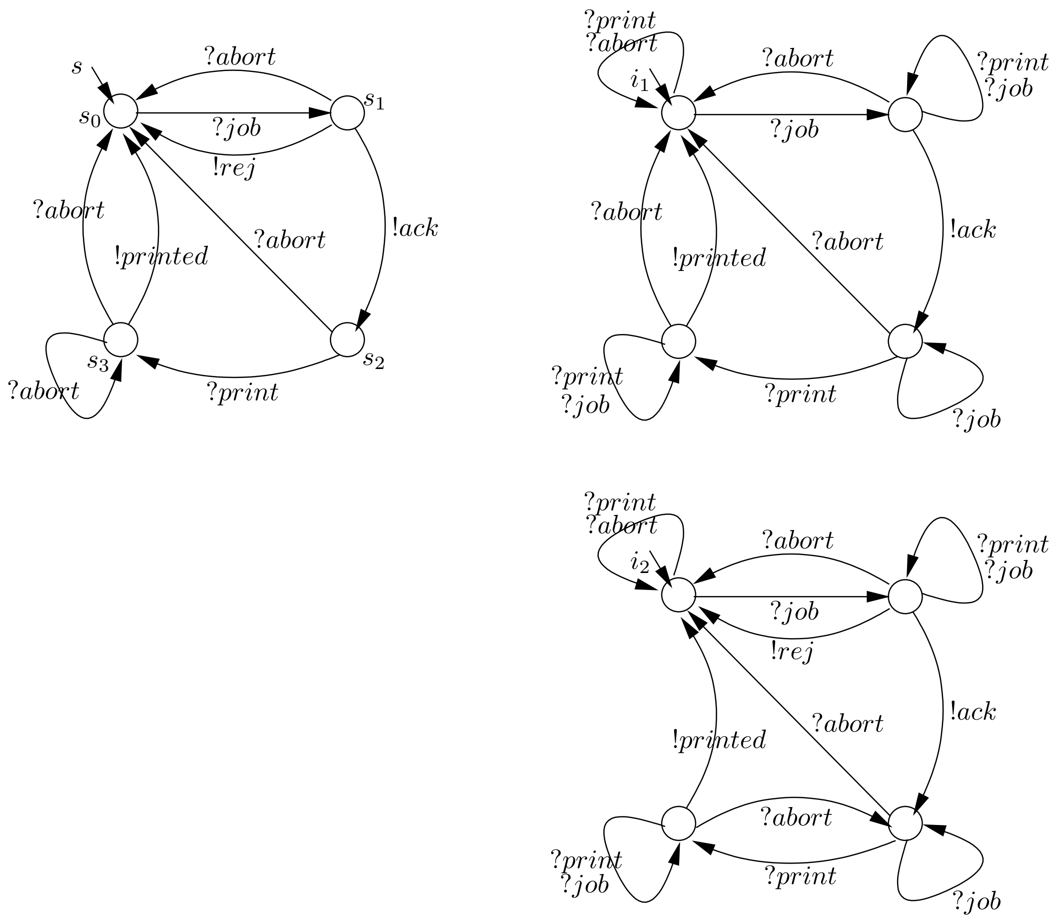


Figure 4: Models of printers, with $L_I = \{?job, ?print, ?abort\}$ and $L_U = \{!ack, !rej, !printed\}$.

□

c. Consider **uioco** as implementation relation:

$$
\begin{array}{lcl}
Utraces(s) & =_{\text{def}} & \{\ \sigma \in Straces(s) \ \mid\ \forall \sigma_1, \sigma_2 \in L_\delta^*,\ a \in L_I :\\
& & \qquad \sigma = \sigma_1 \cdot a \cdot \sigma_2 \text{ implies not } s\ \textbf{after}\ \sigma_1\ \textbf{refuses}\ \{a\}\ \ \}\\
i\ \textbf{uioco}\ s & \Longleftrightarrow_{\text{def}} & \forall \sigma \in Utraces(s) :\quad out(\, i\ \textbf{after}\ \sigma\,) \ \subseteq\ out(\, s\ \textbf{after}\ \sigma\,)
\end{array}
$$

Give a trace of s that is element of $Straces(s)$ but not of $Utraces(s)$.

Answer
Consider the trace $\sigma = ?job \cdot !ack \cdot ?print \cdot ?abort \cdot ?abort$.
Then $s \stackrel{\sigma}{\Longrightarrow}$, so $\sigma \in Straces(s)$.

Now split σ into $\sigma_1 = ?job \cdot !ack \cdot ?print \cdot ?abort$, $a = ?abort$, and $\sigma_2 = \epsilon$. Then $\sigma = \sigma_1 \cdot a \cdot \sigma_2$ and s **after** σ_1 **refuses** $\{a\}$, so $\sigma \notin Utraces(s)$. □

d. Consider the trace $?job \cdot !ack \cdot ?print \cdot ?abort$. How can you observe that the user was 'too slow' to abort the printing? How can you observe that the user was 'fast enough' to abort the printing?

Answer
Consider this trace $?job \cdot !ack \cdot ?print \cdot ?abort$, execute it, and observe the outputs after it.

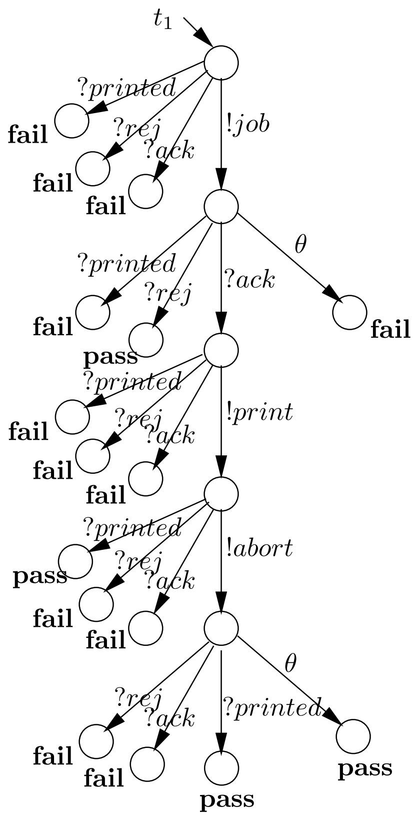


Figure 5: Test case t_1.

Then $out(\, s\ \mathbf{after}\ ?job{\cdot}!ack{\cdot}?print{\cdot}?abort\,)) = \{!printed, \delta\}$.
If $!printed$ is observed as output then printing was not aborted and the user was 'too slow';
If δ is observed as output then printing was aborted and the user was 'fast enough'. $\square$

e. The implementation i_1 has a feature that auto-repairs not well-formed jobs. Therefore, the $!rej$ output is not used. Moreover, i_1 has an AI-based module which takes care that the $?abort$ is always 'fast enough' to abort the printing.

Is the implementation i_1 a **uioco**-correct implementation of s, and why?

Answer
i_1 **uioco** s holds:
the traces where outputs differ, are:
$out(\, i_1\ \mathbf{after}\ ?job\,) = \{!ack\} \subseteq \{!ack, !rej\} = out(\, s\ \mathbf{after}\ ?job\,)$, and
$out(\, i_1\ \mathbf{after}\ ?job{\cdot}!ack{\cdot}?print{\cdot}?abort\,) = \{\delta\}$
$\subseteq \{!printed, \delta\} = out(\, s\ \mathbf{after}\ ?job{\cdot}!ack{\cdot}?print{\cdot}?abort\,)$.
Note that $?job{\cdot}!ack{\cdot}?print{\cdot}?abort{\cdot}?abort \notin Utraces(s)$. $\square$

f. Implementation i_2 also has the AI-based module so that $?abort$ is always 'fast enough'. In i_2, however, $?abort$ after $?print$ does not throw away the job, by going back to the initial state, but goes back to the state before $?print$, so that the user does not have to re-submit the job, when she wants to print it later.

Is the implementation i_2 a **uioco**-correct implementation of s, and why?

Answer
i_2 $\not\!\!\mathbf{uioco}$ s:
$out(\, i_2 \;\mathbf{after}\; ?job{\cdot}!ack{\cdot}?print{\cdot}?abort{\cdot}\delta{\cdot}?job\,) = \{\delta\}$
$\not\subseteq \{!ack, !rej\} = out(\, s \;\mathbf{after}\; ?job{\cdot}!ack{\cdot}?print{\cdot}?abort{\cdot}\delta{\cdot}?job\,)$. □

g. Figure 5 shows the test case t_1 for the *printer*. Test case t_1 aims at testing the output of $?abort$ after $?print$. Give the test run(s) and verdict of applying t_1 to implementation i_2.

Answer

$$t_1 \rceil\!| \; i_2 \xRightarrow{job\cdot rej} \mathbf{pass} \rceil\!| \; i_{2_0}$$
$$t_1 \rceil\!| \; i_2 \xRightarrow{job\cdot ack\cdot print\cdot printed} \mathbf{pass} \rceil\!| \; i_{2_0}$$
$$t_1 \rceil\!| \; i_2 \xRightarrow{job\cdot ack\cdot print\cdot abort\cdot\theta} \mathbf{pass} \rceil\!| \; i_{2_2}$$

All test runs pass, so i_2 **passes** t_1. □

h. Can test case t_1 be generated from s with the **uioco**-test generation algorithm, and why?

Answer
Yes, t_1 can be generated from s with the **uioco**-test generation algorithm, using the following steps of the algorithm:

1. initially in state set $\{s_0\}$ of s, choose input $?job$ going to state set $\{s_1\}$; all outputs in $\{s_0\}$ lead to **fail**;
2. in $\{s_1\}$, choose to observe outputs; output $!rej$ is allowed and leads to $\{s_0\}$, and output $!ack$ is allowed and leads to $\{s_2\}$;
3. after output $!rej$ choose the test case **pass**;
4. after output $!ack$ choose to continue with input $?print$ going to $\{s_3\}$; from $\{s_2\}$, all outputs lead to **fail**;
5. from $\{s_3\}$, choose to provide next input $?abort$; if output $!printed$ is observed before $?abort$ is provided, this leads to $\{s_0\}$; if $?abort$ is accepted the new state set is $\{s_0, s_3\}$; outputs $!rej$ and $!ack$ are not allowead and lead to **fail**;
6. after output $!printed$, in $\{s_0\}$, choose test case **pass**;
7. after having provided input $?abort$, in state set $\{s_0, s_3\}$, choose to observe outputs; outputs $!rej$ and $!ack$ are not allowed and lead to **fail**; output $!printed$ is allowed, and leads to state set $\{s_0\}$; also no output, i.e., quiescence, is allowed and is observed through θ, leading to state set $\{s_0\}$;
8. after both $!printed$ and θ, choose the test case **pass**;
9. the result is the test case t_1 of Fig 5. □

i. Show that test case t_1 is not exhaustive for specification s.

Answer
Exhaustiveness of a test suite T means that all erroneous implementations are detected:

$$\forall i \in \mathcal{IOTS} : \; i \;\mathbf{passes}\; T \text{ implies } i \;\mathbf{uioco}\; s$$

Thus, to show that $\{\, t_1 \,\}$ is not exhaustive, we have to show that there exists an implementation $i \in \mathcal{IOTS}$ that passes $\{\, t_1 \,\}$, yet is not correct: i $\not\!\!\mathbf{uioco}$ s.

From the previous questions we have that i_2 $\not\!\!\mathbf{uioco}$ s and i_2 **passes** t_1. So, i_2 is an implementation that shows that $\{\, t_1 \,\}$ is not exhaustive, □

Grading: Assignment 3

a	b	c	d	e	f	g	h	i	points
3	2	4	3	4	4	3	3	4	max 30

4 Model Learning

In 2012, Arjan Blom, then a student at Radboud University, performed a security analysis of the E.dentifier2 system of the ABN AMRO bank, in which customers use a USB-connected device — a smartcard reader with a display and numeric keyboard — to authorise transactions with their bank card and PIN code. Arjen found a security vulnerability in the E.dentifier2 that was so serious that he even made it to the evening news on Dutch national TV. He did not use systematic testing techniques to find this vulnerability, but in 2014 Erik Poll and colleagues showed that black-box testing of FSMs and model learning could easily reveal the problem with the E.dentifier2. This assignment is based on the FSM models described by Erik Poll et al.

Figure 6 shows the FSM S that specifies the behavior of the E.dentifier2. There are three states $\{q_0, q_1, q_2\}$, five inputs $\{C, D, G, R, S\}$, and four outputs $\{C, L, T, OK\}$. Note that we use commas to indicate multiple transitions. For instance, in S there is both a transition $q_1 \xrightarrow{C/OK} q_1$ and a transition $q_1 \xrightarrow{R/T} q_1$.

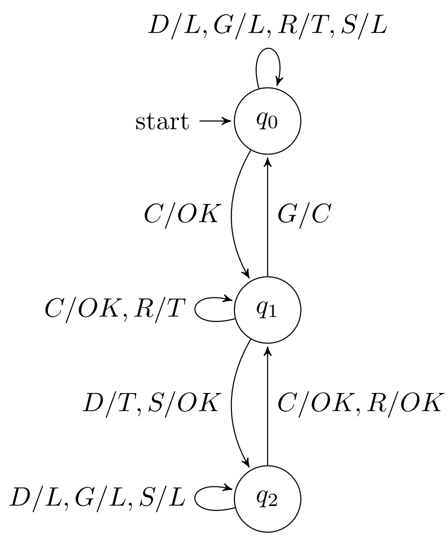


Figure 6: FSM S that specifies the required behavior of the E.dentifier2.

a. Give an access sequence set for S.

b. Give a distinguishing sequence for S or show that this does not exist.

c. Give a UIO for each of the three states of S or show that these do not exist.

d. Give a characterization set for S.

e. Give a 0-complete test suite T for S that is minimal in the sense that when any test in T is omitted it is no longer 0-complete. Explain why your test suite is minimal.

Figure 7 shows the FSM model M for the faulty implementation of the E.dentiefier2.

f. Give a test from your test suite T (if any) that demonstrates that implementation M does not satisfy specification S.

g. Describe a test suite that would reveal for any implementation FSM with at most four states whether or not it satisfies specification S.

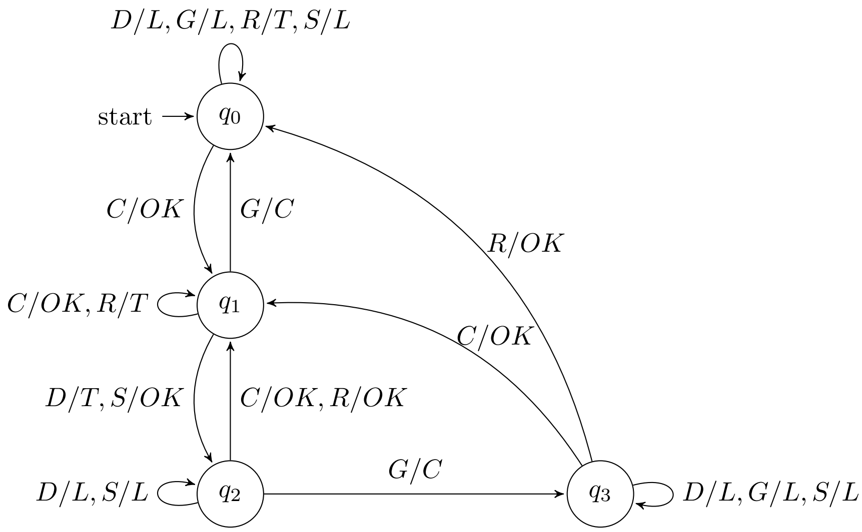


Figure 7: FSM M that describes the faulty implementation of the E.dentifier2.

h. Describe the details of a run of either L^* or $L^\#$ (the choice is yours!) when used to learn specification S. You may also select the counterexamples provided by the teacher.

In the case of L^*, describe the initial rows and columns of the observation table, the specific reason for adding additional rows or columns, all intermediate hypotheses and corresponding counterexamples, and the final table.

For $L^\#$, describe the specific sequence of rule applications to extend the observation tree, all intermediate hypotheses and corresponding counterexamples, and the final observation tree.

Solutions and Correction Guidelines.

a. (3pts) An access sequence set for S is $A = \{\epsilon, C, CD\}$. An alternative access set is $\{\epsilon, C, CS\}$

Grading: Deduct 2pts if ϵ is not included.

b. (3pts) Sequence DR is a distinguishing sequence for S since $\lambda(q_0, D\ R) = L\ T$, $\lambda(q_1, D\ R) = T\ OK$, and $\lambda(q_2, D\ R) = L\ OK$. Sequences GR, RG, RD, SR, RS and DGR are other examples of distinguishing sequences.

c. (3pts) By consequence, sequence DR is also an UIO for each of the three states of S. But shorter UIOS exist for q_1 and q_2: D, G and S are UIO's for q_1, and R is an UIO for q_2.

d. (3pts) Moreover, $\mathcal{C} = \{DR\}$ is a characterization set for S.

e. (6pts) Let $I = \{C, D, G, R, S\}$ be the set of inputs.

- As explained during the lecture, $\mathcal{T} = A \cdot I^{\leq 1} \cdot \mathcal{C}$ is a 0-complete test suite for S. This test suite comprises the following 18 tests:

$$
\begin{aligned}
&DR, \cancel{CDR}, \cancel{CDDR},\\
&\cancel{CDR}, DDR, GDR, RDR, SDR,\\
&CCDR, CDDR, CGDR, CRDR, CSDR,\\
&CDCDR, CDDDR, CDGDR, CDRDR, CDSDR
\end{aligned}
$$

Since they are prefixes of other tests, we can omit the three tests that are striked through. This gives us a 0-complete test suite with 15 tests.

- Ten tests from $\mathcal{T}$ cannot be omitted as the are the only test that visit a certain transition. If we would omit these tests, then the test suite might not discover a possible output fault:
  1. GDR is the only test that visits the outgoing G-transition of q_0.
  2. SDR is the only test that visits the outgoing S-transitions of q_0.
  3. $CCDR$ is the only test that visits the outgoing C-transition of q_1.
  4. $CGDR$ is the only test that visits the outgoing G-transition of q_1.
  5. $CRDR$ is the only test that visits the outgoing R-transition of q_1.
  6. $CSDR$ is the only test that visits the outgoing S-transition of q_1.
  7. $CDCDR$ is the only test that visits the outgoing C-transition of q_2.
  8. $CDGDR$ is the only test that visits the outgoing G-transition of q_2.
  9. $CDSDR$ is the only test that visits the outgoing S-transition of q_2.
  10. CDD and $CDRDR$ cannot both be omitted because these are the only tests that visit the outgoing R-transition from q_2.
- Three tests from $\mathcal{T}$ cannot be omitted since then the test suite might no longer detect a possible transition fault:
  1. DDR cannot be omitted because otherwise the FSM obtained from S by redirecting the D-transition from q_0 to q_1 would pass all tests.
  2. RDR cannot be omitted because otherwise the FSM obtained from S by redirecting the R-transition from q_0 to q_2 would pass all tests.
  3. $CDDDR$ cannot be omitted because otherwise the FSM obtained from S by redirecting the D-transition from q_2 to q_0 would pass all tests.
- Consider the test suite $\mathcal{T}'$ that only contains the 13 tests for which we have shown above that they cannot be omitted (we picked $CDDDR$ rather than $CDDR$, similar argument applies if we pick $CDDR$):

$$\begin{gathered}DDR, GDR, RDR, SDR,\\ CCDR, CGDR, CRDR, CSDR,\\ CDCDR, CDDDR, CDGDR, CDRDR, CDSDR\end{gathered}$$

  Clearly, $\mathcal{T}'$ is minimal. In order to see that it is also 0-complete, consider Figure 8, which shows the observation tree obtained if the SUL passes all these 13 tests (leaving out the distinguishing sequences DR at the end). Note that the three magenta states are pairwise apart and constitute a basis. Furthermore note that all frontier states are identified by distinguishing sequence DR. Thus we may construct an hypothesis as in the $L^{\#}$ algorithm. This hypothesis is equivalent to the specification FSM S. Now the following theorem from the paper that introduces the $L^{\#}$ algorithm directly implies that $\mathcal{T}'$ is 0-complete:

  Theorem 3.7. *Suppose $\mathcal{T}$ is an observation tree for the Mealy machine $\mathcal{M}$ of the SUL. Suppose each basis state has outgoing transitions for every input, all states in the frontier are identified, and the number of states in the basis is equal to the number of states of $\mathcal{M}$. Let $\mathcal{H}$ be the hypothesis constructed from $\mathcal{T}$. Then $\mathcal{H}$ and $\mathcal{M}$ are equivalent.*

Grading: 2 pts for a 0-complete test suite, 2pts if redundant prefixes are removed, 2pts for convincing argument for minimality. The question about minimality is by far the hardest question of the exam. Different solutions are possible, but the solution presented here shows a nice correspondence between model learning and model-based testing.

f. (3pts) Test $CDGDR$ from test suite $\mathcal{T}$ demonstrates that implementation M does not satisfy specification S, as the output for the G input should be L, but in M a C is generated.

Grading: 2 pts if no test suite was given in item e, but a correct test is provided that demonstrates the problem.

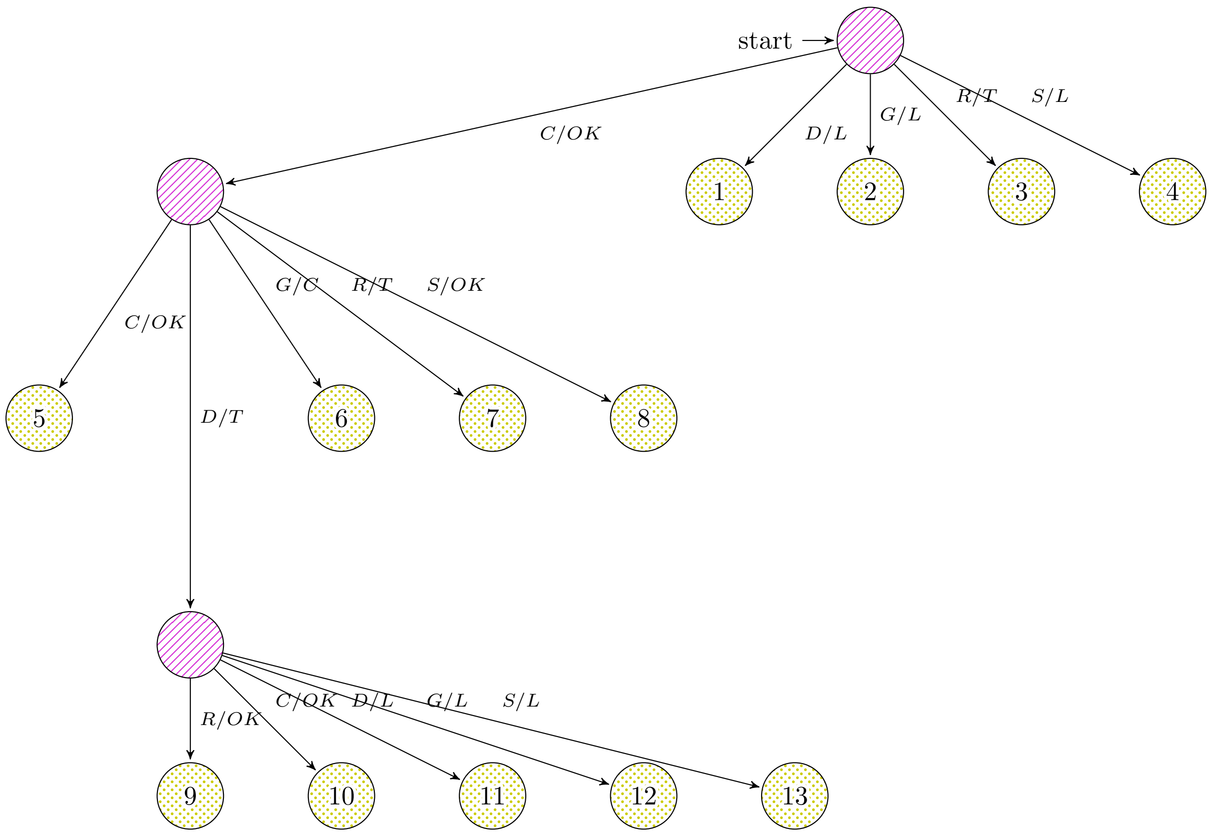


Figure 8: Observation tree obtained if SUT passes all 13 tests from $\mathcal{T}'$ (we have omitted the distinguishing sequence DR from the leaves)

g. (3pts) Test suite $\mathcal{T} = A \cdot I^{\leq 2} \cdot \mathcal{C}$ would reveal for any implementation FSM with at most four states whether or not it satisfies specification S.

h. (16pts) Describe the details of a run of either L^* or $L^{\#}$ (the choice is yours!) when used to learn specification S. You may also select the counterexamples provided by the teacher.

Solution L^* The L^* algorithm constructs an observation table by performing the steps below:

1. The set $\mathcal{U}$ of prefixes is initialized to $\{\epsilon\}$ and the set $\mathcal{V}$ of suffixes to $\{C, D, G, R, S\}$. The L^* algorithm then poses 30 output queries to fill the following table:

	C	D	G	R	S
ϵ	OK	L	L	T	L
C	OK	T	C	T	OK
D	OK	L	L	T	L
G	OK	L	L	T	L
R	OK	L	L	T	L
S	OK	L	L	T	L

2. The table is not closed since the rows for prefixes ϵ and C are not equivalent. Thus set $\mathcal{U}$ is extended to $\{\epsilon, C\}$. One letter extensions of the new prefix are added as rows to the table, and the learner poses 25 additional output queries to fill the extended table:

	C	D	G	R	S
ϵ	OK	L	L	T	L
C	OK	T	C	T	OK
D	OK	L	L	T	L
G	OK	L	L	T	L
R	OK	L	L	T	L
S	OK	L	L	T	L
CC	OK	T	C	T	OK
CD	OK	L	L	OK	L
CG	OK	L	L	T	L
CR	OK	T	C	T	OK
CS	OK	L	L	OK	L

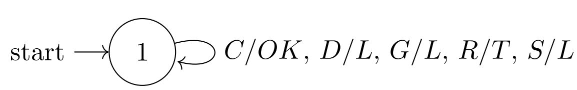
3. The table is still not closed since the rows for prefix CD is different from those for prefixes ϵ and C. Thus set $\mathcal{U}$ is extended to $\{\epsilon, C, CD\}$. One letter extensions of the new prefix are added as rows to the table, and the learner poses 25 additional output queries to fill the extended table:

	C	D	G	R	S
ϵ	OK	L	L	T	L
C	OK	T	C	T	OK
CD	OK	L	L	OK	L
D	OK	L	L	T	L
G	OK	L	L	T	L
R	OK	L	L	T	L
S	OK	L	L	T	L
CC	OK	T	C	T	OK
CG	OK	L	L	T	L
CR	OK	T	C	T	OK
CS	OK	L	L	OK	L
CDC	OK	T	C	T	OK
CDD	OK	L	L	OK	L
CDG	OK	L	L	OK	L
CDR	OK	T	C	T	OK
CDS	OK	L	L	OK	L

4. Now the table is closed and consistent. We can construct a hypothesis with states ϵ, C and CD, and transitions as specified in the table. The resulting FSM is equivalent to S (q_0 corresponds to ϵ, q_1 to C and q_2 to CD), so the teacher will tell the learner that the hypothesis is correct.

Solution $L^{\#}$ The $L^{\#}$ algorithm constructs the observation tree $\mathcal{T}$ of Figure 10 by performing the steps below. Note that this is just one possible run of the algorithm as rules may be applied in different orders and the teacher may provide other counterexamples.

1. The initial observation tree has a single state 1, which constitutes the basis.
2. The extension rule is applied five times to explore the outgoing transitions of state 1, for inputs C, D, G, R and S, leading to new frontier states 2, 3, 4, 5 and 6, respectively.
3. Since the frontier has no isolated states and the basis is complete, the learner applies rule (R4) and submits a first hypothesis to the teacher:



Suppose a helpful teacher returns counterexample CDR. Then the observation tree is extended with transitions from state 2 to a new state 7, and from there to new state 8. Since the counterexample leads to a conflict on the frontier, counterexample processing finishes immediately.

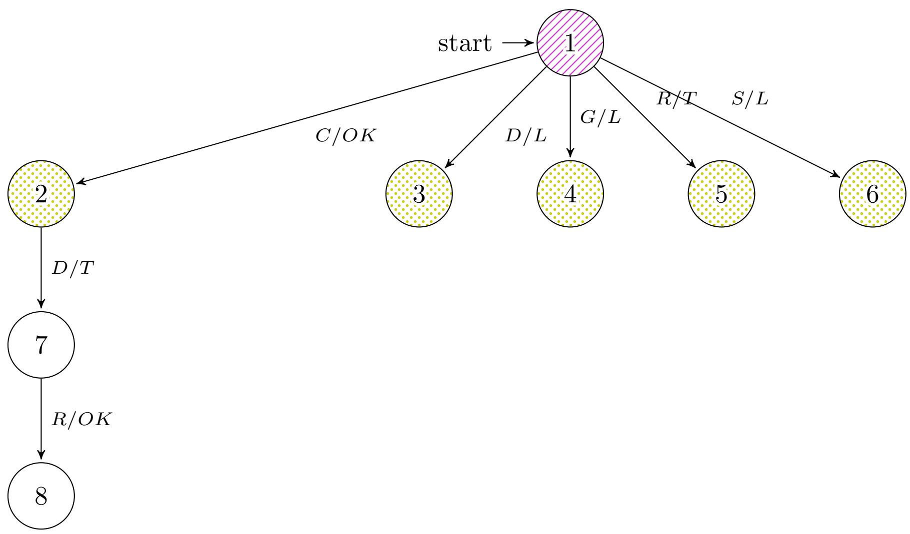


Figure 9: Observation tree after receipt of first counterexample

4. Witness D shows that states 1 and 2 are apart, and thus state 2 is promoted to the basis and state 7 becomes a frontier state.
5. The extension rule is applied to explore the outgoing transitions of state 2 for the remaining inputs C, G, R and S, leading to new frontier states 9, 10, 11 and 12, respectively.
6. Now we observe that witness R shows that state 7 is apart from states 1 and 2. Therefore, state 7 is promoted to the basis.
7. The extension rule is applied to explore the outgoing transitions of state 7 for the remaining inputs C, D, G and S, leading to new frontier states 13, 14, 15 and 16.
8. The learner repeatedly applies the identification rule to identify the 13 frontier states, using witnesses D and R. In order to identify a frontier state at least 1 and at most 2 output queries are needed.
9. The resulting hypothesis $\mathcal{H}_2$ is equivalent to $\mathcal{M}$.

Thus $L^\#$ needs (at most) 40 output queries and 2 equivalence queries to learn the model. However, for this we assume a helpful teacher that provides an informative counterexample. Note that L^* needs at least twice as many output queries, but no counterexample provided by the teacher, in order to learn this FSM.

Grading: Assignment 4

a	b	c	d	e	f	g	h	points
3	3	3	3	6	3	3	16	max 40

The End

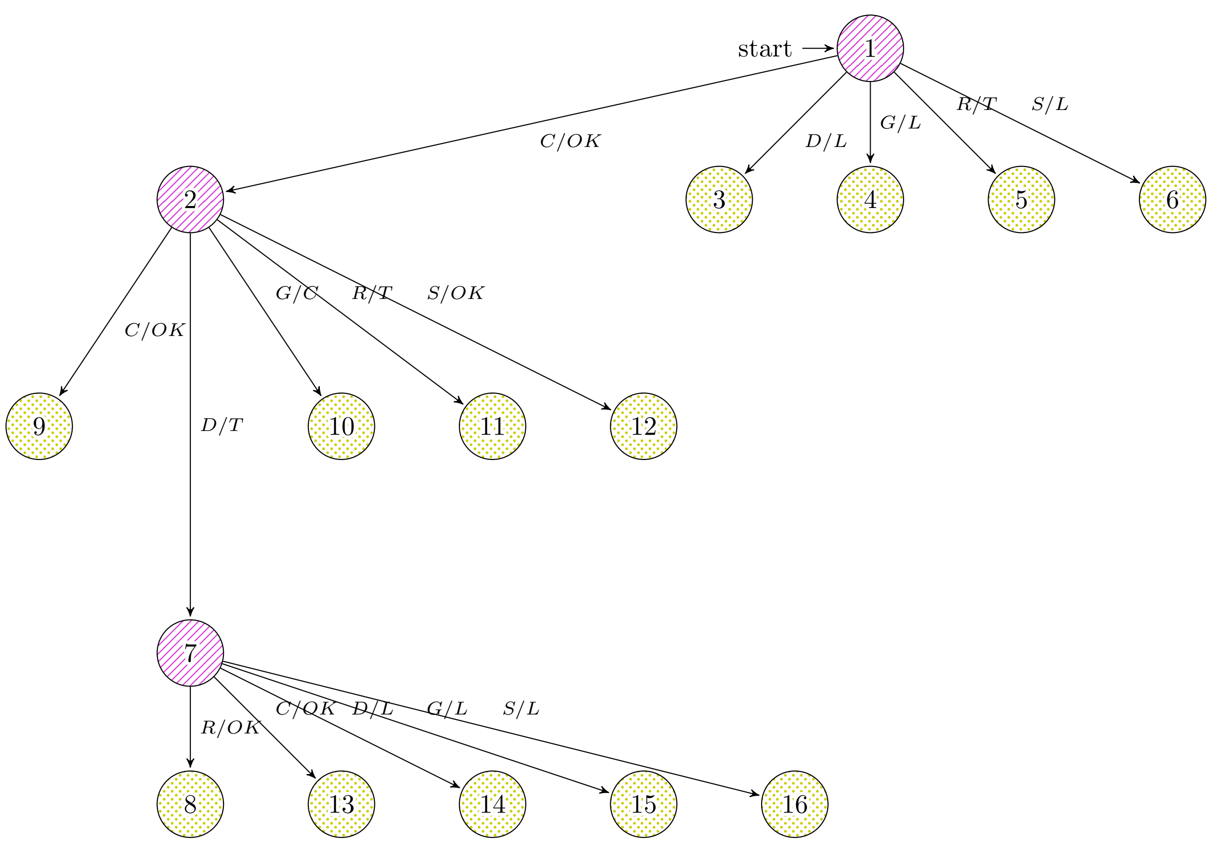


Figure 10: Final observation tree (outgoing D and R transitions of frontier states not drawn)