Chapter 4

Economics of Testing

4.1 Introduction

Until now, we have concentrated on *what* is testing, and to some extent *how* it can be carried out in a formal conformance testing setting. Yet, another as important question to be asked is: *why* to test?

Testing, eventually, is an economic activity. There is no such thing as free testing. Testing always costs something either directly or indirectly: money, working time, other resources. Because there is a cost, there must be a pay-off. The pay-off is reduced risk of a system malfunctioning after it has been taken into production use.

But what is the cost of that risk? It depends on what is the utility of the system in production use. If the system is useless, it does not matter if it crashes or otherwise malfunctions. On the other hand, if the system is life-critical, any risk of failure should be avoided.

Other systems could be not mission-critical, but could still have a very high pay-off when used and high cost when not working. For example, the Windows desktop environment is not usually life-critical. But if the desktop would crash once a week, destroying approximately half an hour's work, a 1000–person corporation would lose 13 man-years per annum—clearly a cost to be taken into account.

Therefore, it seems that we should be able to compare the following costs:

- The cost of testing,
- the cost of system crash (life-critical systems), and
- the indirect cost of time lost due to a malfunctioning system.

To achieve this, we will attach a cost to every test step—for our purposes it suffices to assume a fixed cost per a test step, because test steps can be defined rather freely. Then, we will attach a pay-off to every "use step"; a use step is just a test step interpreted in the context of production use rather than in the context of testing. Third, there will be a cost attached to every use step that results in malfunctioning.[1]

Then the average usefulness of a system can be characterized by

$$\text{use pay-off} - \frac{\text{cost of crash}}{\text{expected n.o. steps before failure}}. \tag{4.1}$$

But where does testing come in? The purpose of testing is to increase the expected length of correct runs. The key point is that *this is possible without modifying the system under test*. From one viewpoint, certainly, if the SUT is not modified, the average length of correct executions cannot change. And this is true. But from the viewpoint of *deciding* whether a system can be shipped or not, the *expected* failure rate can be lowered by demonstrating—by testing—that the system behaves correctly.

A more mathematical approach to these issues is the main subject of this chapter.

[1]It must be noted that this model is not complete in the sense that programs have often bugs from which it is possible to recover. Hence, even though the programs do not "crash", they work incorrectly "for a while". This situation is problematic to analyze within our framework, because an incorrect execution (trace) cannot be turned into a correct execution (trace) by continuing it. We will assume that after a system works incorrectly, it will be "reset" in some way. This is a drawback from the generality point of view, but it does not make the analysis here less relevant.

4.2 Introduction to Expected Utility Theory

We have at our hand a problem that involves two basic quantities: money (we can consolidate all resource use into the use of money), and incertainty, which we will model with probabilities. Basically we have two incompatible things, and to combine them we need some sort of a theory. The theory that we will use is known as "expected utility theory" in economics; it is probably mostly the same as "rational decision theory"; the name does not matter here, because the basics of the theory are very simple.

Expected utility theory stems from the following consideration. Suppose that you must make a decision between two choices. If you perform rationally, you will choose that choice that is more useful for you. In other words, you make a choice that yields better *utility*. If we assume that it is always possible to find one best alternative from a set of alternatives, then it is easy to postulate that there is something general known as "utility" that can be presented as a real number and that can be attached to every potential alternative, or the outcome of such an alternative.

Utility does not need to have a named unit; it is an abstract measure of usefulness.

A *lottery* is now a probability distribution over a set of *alternatives* (whatever they are). Let us denote a set of alternatives by A. Then

$$\mathscr{L} : A \to [0,1] \tag{4.2}$$

is a lottery iff

$$\sum_{\alpha \in A} L(\alpha) = 1. \tag{4.3}$$

Every alternative α has a certain utility, marked by $u(\alpha)$. The expected utility theorem assumes that the utility of the lottery $\mathscr{L}$ itself is given by

$$u(\mathscr{L}) = \sum_{\alpha \in A} \mathscr{L}(\alpha) u(\alpha). \tag{4.4}$$

This is a "utility function" for lotteries. (If you want to make a good impression, you can call it a *von Neumann–Morgenstern expected utility function*.) Thus, *the utility of a lottery is its statistically expected utility*. The normal theories of statistics came into play, importantly linearity of expectation, which yields in our context the linearity of [expected] utility.

Example 4.1. For example, a rational agent should be indifferent choosing between utility of 1, and a lottery that yields utilities 0 and 2, both with probability $\frac{1}{2}$.

4.2.1 General Nonlinearity of Monetary Value

The expected utility theory assumes linearity of utility. One of the most prevalent real units for utility is money. At least in business life, almost anything can be consolidated (sometimes in very surprising ways) into money. But the problem is that people do not take monetary value as linear.

Consider the following two alternatives: (1) You get immediately million euros, certainly and for free. (2) You take part in a lottery, where you have the chance of winning one hundred million euros, but the probability of winning is only 2%.

If we assume that "utility = money", the utility of the first alternative is 10^6€, and the utility of the second alternative is $10^8 \times 0.02 = 2 \times 10^6$€. A rational agent should choose the alternative number two. Still, at least I would choose alternative number one. As a matter of fact, I believe many others would also prefer (1).

This is an effect of the non-linear utility of money, a general phenomenon. For a poor guy, the difference between owing nothing and one million is probably even greater than the difference between owing one million and one hundred million euros. Every additional euro is worth less than the previous ones.

4.2.2 Assumed Linearity

However, in our context we assume that the utility of money *is linear*. Therefore, we *can* state—for our purposes—that "utility = money". We can thus use euro as the unit of utility, if for nothing else, then for the concreteness' sake.

4.3 Simple Economic Model of Testing

4.3.1 Failing Probabilities

Let i be an implementation and s a testing strategy (in this chapter, testing strategies will be also used as models of general use and interaction). For every natural number n and a trace T, let $P_n[i,s,T]$ denote the probability that the trace T has been produced *exactly in n test steps* when the strategy s is executed against the implementation i. Because for every trace there is a unique way to produce it as a sequence of test steps, this is easily defined.

DEFINITION 4.2. $P_n[i,s,T]$ is the maximum over all sequences $\epsilon = T_0, \ldots, T_n = T$ of the expression

$$P_n[i,s,T] = \prod_{i\in[1,k-1]} \xi(i,s,T_i)[T_{i+1}]. \tag{4.5}$$

Note that this definition is identical with (2.14), the only difference being that we take the product only over trace sequences of a definite length n.

We have now a series of trace probability distributions, indexed by natural numbers. These distributions correspond to traces that we get when we continue execution, eventually arbitrarily far. We can calculate the probability that after nth step, the produced trace is invalid:

DEFINITION 4.3. Let i be an implementation, s a testing strategy and S a specification. The probability that a trace produced after n test steps is an invalid one is given by

$$F_n[i,s,S] = \sum_{T\notin\mathbf{Tr}(S)} P_n[i,s,T]. \tag{4.6}$$

Note that because $T \preceq T'$ and $T \notin \mathbf{Tr}(S)$ implies $T' \notin \mathbf{Tr}(S)$ (valid trace sets are prefix-complete), we get

$$\forall n \in \mathbb{N} : F_n[i,s,S] \leq F_{n+1}[i,s,S]. \tag{4.7}$$

Hence, the probability that the current execution is invalid one cannot decrease when time goes on. This is correct.

The expected length of correct execution can now be calculated. Suppose i, s and S have been fixed. The probability of getting a failure on the first step (i.e. producing a correct trace of length zero steps) is F_1 (note that $F_0 = 0$ always). The probability of producing a correct trace of exactly length one is similarly $(F_2 - F_1)$. The probability of producing a correct trace of exactly length two is, obviously, $(F_3 - F_2)$. And so on. In general, the probability $F_i - F_{i-1}$ is the probability that we produce a trace that fails exactly on step i.

Assuming $lim_{n\to\infty} F_n = 1$, we can compute the expected number of steps before first failure as

$$E = \sum_{i>0} (i-1)(F_i - F_{i-1}), \tag{4.8}$$

if this series converges. If it converges, it is equal to

$$E = \sum_{i>0} (1 - F_i)\,. \tag{4.9}$$

Namely, summing up the series (4.8) upto k, we get

$$\left(\sum_{0<i\leq k} -F_i\right) + kF_{k+1}. \tag{4.10}$$

By assumption, F_n tends to 1 when n tends to infinity; hence, in the limit this becomes

$$\left(\sum_{0<i\leq k} -F_i\right) + k = \sum_{0<i\leq k} (1 - F_i)\,. \tag{4.11}$$

If $lim_{n\to\infty} F_n$ is not 1, the expected length of correct execution cannot be a number but is infinite. The reason is that there is the probability of $1 - lim_{n\to\infty} F_n$ that a produced trace continues as correct trace *ad infinitum*, and “multiplying” this infinity with a finite probability still yields infinite expectation. (On the other hand, $lim_{n\to\infty} F_n = 1$ does not imply convergence; the expected length of correct execution can still be infinite. For example, with $F_i = 1 - i^{-1}$, (4.9) becomes the Harmonic Series.)

4.3.2 Absolutely Correct Systems

Let us say that an implementation i is *absolutely correct* if for all $s \in \mathbf{CT}(S)$, $\lim_{n\to\infty} F_n[i,s,S] = 0$. The system never fails against any correct test or use strategy.

Recall that a *failure model* is a mapping $\mu : \mathbb{S} \to (\mathbb{I} \to [0,1])$, which maps every specification onto a probability distribution over implementations.

For a given specification S, denote

$$A(S) = \{i \mid i \in \mathbb{I} \wedge \forall s \in \mathbf{CT}(S) : \lim_{n\to\infty} F_n[i,s,S] = 0\}. \tag{4.12}$$

This is the set of absolutely correct implementations for S.

Suppose we are given failure model μ such that, for a given specification S, $\mu(S) = \psi$, ψ being a probability distribution.The probability that an implementation of S—picked out from the space of all implementations according to the distribution ψ—is absolutely correct is

$$\sum_{i\in A(S)} \psi[i]. \tag{4.13}$$

This is the *a priori absolute correctness probability* for the specification S, assuming the failure model μ. Of course, it is not about the correctness of S; it is about the correctness of an unknown *implementation* of S.

It is time to bring forth a key point of testing. Testing may detect errors and misfeatures. If this happens, there is usually a tendency to fix the detected errors; even if the detected errors will not be fixed, testing has created some useful value. But even if testing does *not* find any errors, certainly the testing has been of some value. But where is that value?

ψ, the probability distribution given by the failure model, describes assumptions about the potential implementations before any further information has been obtained. But testing activity is bound to usually change the set of potential implementations (and implementation faults, if you wish). After testing, the best understanding about the current implementation is not described by ψ, which is the *a priori* information, but by another probability distribution on implementations, $\hat{\psi}$, the *a posteriori*, after-testing, distribution.

Bayes' Rule:

$$P(A|B) = P(B|A)\frac{P(A)}{P(B)}.$$

As usual, this distribution could be computed by the Bayes' Rule.

Suppose that we have tested a system with strategy s and against specification S. We have produced trace T, such that $\mathbf{verdict}(T,S) = \text{PASS}$. (There were thus no errors). For any particular implementation i, the probability that i could produce this trace is given by $P[i,s,T]$. Hence, the *a priori* probability for observing this trace was

$$\sum_{i\in\mathbb{I}} \psi[i]P[i,s,T]. \tag{4.14}$$

On the other hand, the *a priori* probability for the implementation i itself was $\psi[i]$. By Bayes' Rule, we get the *a posteriori* probability for i being the unknown implementation as

$$\hat{\psi}[i] = P[i,s,T]\frac{\psi[i]}{\sum_{i^*\in\mathbb{I}} \psi[i^*]P[i^*,s,T]}. \tag{4.15}$$

A couple of notes are due:

- If the *a priori* probability for i is zero, then is the *a posteriori* also, because $\psi[i]$ is a factor of the right-hand side of (4.15).
- Similarly, if an implementation i cannot produce the trace T at all, the *a posteriori* probability becomes zero.
- If there are two systems i and i' such that $P[i,s,T] = kP[i',s,T]$ for some k, then

$$\frac{\hat{\psi}[i]}{\hat{\psi}[i']} = k\frac{\psi[i]}{\psi[i']}. \tag{4.16}$$

  The reason is that the denominator in (4.15) is constant for all implementations. Thus, if a system produces the observed trace three times as "often" as another system, the probability of the first system is scaled up by a factor of three with respect to the other system's probability, in the *a posteriori* distribution.

(These all notes are obvious effects of applying Bayes' Rule, but they can function as a cross-checking aid and increase the level of concreteness.)

The *a posteriori* probablity for the absolute correctness of the unknown implementation is

$$\sum_{i \in A(S)} \hat{\psi}[i]. \tag{4.17}$$

The thousand dollars question is whether we could via testing increase the level of trust in our implementation; that is, whether we could choose a testing strategy s that would produce a trace T against the black box so that

$$\sum_{i \in A(S)} (\hat{\psi}[i] - \psi[i]) > 0. \tag{4.18}$$

In general, there is no absolutely affirmative answer to this question. Under some general assumptions, it is possible that we produce a trace T such that $\mathbf{verdict}(T, S) = \textsc{pass}$ but still

$$\sum_{i \in A(S)} (\hat{\psi}[i] - \psi[i]) < 0. \tag{4.19}$$

That is, the level of trust into the system's absolute correctness decreases. Dwelling into this subject deeper is out of the scope of this class; but this is a fact that we may want to be aware of.

Chapter 5

Ioco Testing Theory

In this chapter we make a side step from our general framework, and examine another formal conformance testing theory, which is colloquially known as the "ioco testing theory". It is a formalization of conformance testing by Jan Tretmans *et al*, based on the concept of labelled transition systems (LTS).

5.1 Labelled Transition Systems

A labelled transition system is a formal object, defined as follows.

DEFINITION 5.1. A *labelled transition system* is a tuple $\langle S, L, T, s_0 \rangle$ where S is a countable set of *states* and L is a countable set of *labels*. Letting L_τ denote $L \cup \{\tau\}$, where we assume $\tau \notin L$, $T \subseteq S \times L_\tau \times S$ (known as the *transition relation*), and $s_0 \in S$ is the *initial state*.

When an LTS is implied by context, we write $s \xrightarrow{\mu} s'$ to denote that $\langle s, \mu, s' \rangle \in T$, i.e. that there is a transition from s to s' with the action μ. Note that μ is either a label (member of L), or the special symbol τ, which denotes "internal action".

Furthermore, we write $s \xRightarrow{\lambda} s'$ for $\lambda \in L$ iff there exist states $s_1, \ldots, s_n$ such that $s_1 = s$, $s' = s_n$, and for $1 \le k \le n$,

$$s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_k \xrightarrow{\lambda} s_{k+1} \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_n. \tag{5.1}$$

Finally, if $w = \lambda_1 \cdots \lambda_n$ is a word over labels, i.e. a member of the set L^*, we write $s \xRightarrow{w} s'$ iff there exist $s_1, \ldots, s_{n+1}$ such that $s_1 = s$, $s' = s_{n+1}$, and

$$s_1 \xRightarrow{\lambda}_1 s_2 \xRightarrow{\lambda}_2 \cdots \xRightarrow{\lambda}_n s_{n+1}. \tag{5.2}$$

Note that for the empty word over labels ϵ, $s \xRightarrow{\epsilon} s'$ iff there is a path from s to s' via only internal actions (τ-transitions).

5.1.1 Parallel Composition

We will now define a *parallel composition* operation for labelled transition systems. Parallel composition creates a new LTS from two LTSes. The resulting LTS is intuitively obtained by executing the two original LTSes in parallel, synchronizing all other actions but the internal ones (τ's). Let us look on the formal definition.

DEFINITION 5.2. Let $\mathscr{L} = \langle S, L, T, s_0 \rangle$ and $\mathscr{L}' = \langle S', L, T', s'_0 \rangle$ be two LTSes, assuming, without loss of generality, that the set of labels is the same for both of the systems.

The parallel composition of $\mathscr{L}$ and $\mathscr{L}'$ is denoted by $\mathscr{L} \,||\, \mathscr{L}'$, and is defined as follows: the system $\mathscr{L} \,||\, \mathscr{L}'$ is $\langle S \times S', L, T^*, \langle s_0, s'_{,0} \rangle \rangle$ where T^* is the least relation over $(S \times S') \times L_\tau \times (S \times S')$ fulfilling the following:

1. If $\langle s, \tau, t \rangle \in T$ and $s' \in S'$ then $\langle \langle s, s' \rangle, \tau, \langle t, s' \rangle \rangle \in T^*$.
2. If $\langle s', \tau, t' \rangle \in T'$ and $s \in S$ then $\langle \langle s, s' \rangle, \tau, \langle s, t' \rangle \rangle \in T^*$.

3. If $\lambda \in L$, $\langle s, \lambda, t\rangle \in T$ and $\langle s', \lambda, t'\rangle \in T'$, then $\langle\langle s, s'\rangle, \lambda, \langle t, t'\rangle\rangle \in T^*$.

Note how the parallel composition allows for the two original components to execute internal actions so that the other components remains in its current state, but that observable actions (= labels) must be executed so that both components execute simultaneously the same action.

DEFINITION 5.3. For an LTS $\mathscr{L} = \langle S, L, T, s_0\rangle$, denote by traces($\mathscr{L}$) the set of words over L^*

$$\{w \mid \exists s : s_0 \overset{w}{\Longrightarrow} s\}. \tag{5.3}$$

5.1.2 Input-Output Transition Systems

In testing setups it is usually the case that a system cannot prohibit the attempt of sending an input to the system itself. The input could well be illegal, i.e. out of specification, but physically blocking the input so that an external agent would “know” that the input cannot be sent can be assumed impossible. One way to see this is to observe that if an external agent “knows” that an input cannot be sent, or that it is blocked, then some information has been exchanged outside the normal realm of communications. If all communications are modelled explicitly, such a special mechanism does not need to exist.

To reflect this idea on the level of LTSes, there exists a concept of an *input-output transition system*, which is commonly abbreviated as IOTS. An IOTS is an LTS whose label set L has been divided into two partitions: input labels and output labels, and that can always synchronize on any input label. The formal definition comes next.

DEFINITION 5.4. An *input-output transition system* is an LTS $\mathscr{L} = \langle S, L, T, s_0\rangle$ and an understood partition of L into two disjoint partitions L_{in} and L_{out} such that

$$(\exists w \in L^* : s_0 \overset{w}{\Longrightarrow} s) \implies (\forall \alpha \in L_{\text{in}} : \exists s' : s \overset{\alpha}{\Longrightarrow} s'). \tag{5.4}$$

The above equation says that whenever there is a reachable state s within the LTS $\mathscr{L}$, there must exist for every input label a (potentially empty) sequence of internal actions, after which the system can synchronize with the chosen input label.

5.2 Testing Labelled Transition Systems

We move now on to consider how LTSes can be used to implement or represent conformance testing. We will assume here that that there exist a fixed set of labels L, which has been partitioned into L_{in} and L_{out}.

An SUT is now any IOTS over L, with L_{in} being the input labels. A specification is a similar IOTS, with L_{in} again being the set of input labels; it is an “executable” reference implementation. A testing strategy or just tester, on the other hand, is an IOTS whose set of input labels is the set L_{out}: because testers must communicate with SUTs rather than describe their behaviour, the roles of input and output labels must be reversed.

If $\mathscr{T}$ is a tester and $\mathscr{X}$ is an SUT, the “execution” of the tester against the SUT is described by the parallel composed system $\mathscr{T} \,||\, \mathscr{X}$. Similarly, if $\mathscr{S}$ is a specification, the parallel composed systen $\mathscr{T} \,||\, \mathscr{S}$ denotes the hypothetical execution of the tester against the specification, which is here actually an executable reference implementation.

5.2.1 Observations

The notion of conformance or correctness of a system with respect to a specification is developed in the context of the ioco theory as described next. We assume that there exists an universe of observations (denote it by O) with an associated partial ordering $\sqsubseteq$ over observations. Furthermore, there exists a function obs that maps any pair of compatible IOTSes (tester + implementation or tester + specification) to an observation. Then, an SUT $\mathscr{X}$ is correct with respect to a specification $\mathscr{S}$ if and only if for all testers $\mathscr{T}$, it holds that

$$\mathsf{obs}(\mathscr{T}, \mathscr{X}) \sqsubseteq \mathsf{obs}(\mathscr{T}, \mathscr{S}). \tag{5.5}$$

In other words, anything that we can observe when we execute any chosen tester against the SUT $\mathscr{X}$, we could observe if we would execute the same tester against the specification and reference implementation $\mathscr{S}$.

Different structures of the universe of observations yield different notions for conformance. We shall focus on the conformance relation that has been given the name **ioco**.

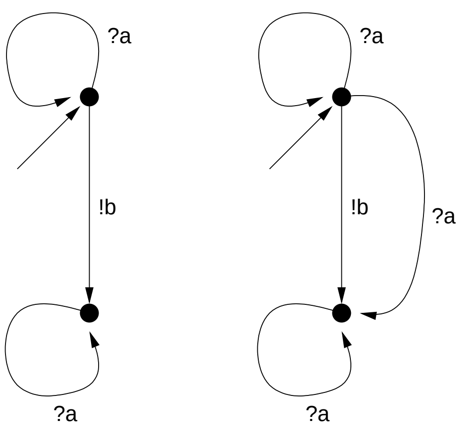

Figure 5.1: Two IOTSes.

5.2.2 Trace Inclusion

The first observation structure that might come into mind is the structure of trace inclusion. Here O is the set of sets of traces, $\sqsubseteq$ is subset relation and obs produces the set of all potential traces of the parallel composition of two systems. Formally, let

- $O = 2^{L^*}$,
- $T \sqsubseteq T'$ iff $T \subseteq T'$, and
- $\mathtt{obs}(\mathscr{I}, \mathscr{J}) = \mathrm{traces}(\mathscr{I} \,||\, \mathscr{J})$.

Consider now an SUT $\mathscr{X}$ and a specification $\mathscr{S}$. Because of the nature of parallel composition, for any two LTSes $\mathscr{L}$ and $\mathscr{L}'$, $\mathrm{traces}(\mathscr{L} \,||\, \mathscr{L}') \subseteq \mathrm{traces}(\mathscr{L})$. Hence, $\mathrm{traces}(\mathscr{X}) \subseteq \mathrm{traces}(\mathscr{S})$ implies that for any tester $\mathscr{T}$, $\mathtt{obs}(\mathscr{T}, \mathscr{X}) \sqsubseteq \mathtt{obs}(\mathscr{T}, \mathscr{S})$. On the other hand, suppose there exists $T \in \mathrm{traces}(\mathscr{X}) \setminus \mathrm{traces}(\mathscr{S})$, i.e. that there exists a trace in the traces of the SUT but that is not a trace of the specification. From that trace we can convert a tester that is capable of reproducing that trace against the implementation. By the argument above, against the specification this trace is not reproducible, and hence $\mathtt{obs}(\mathscr{T}, \mathscr{X}) \not\sqsubseteq \mathtt{obs}(\mathscr{T}, \mathscr{S})$.

Hence, with this structure, conformance corresponds to the inclusion of the set of traces of an SUT in the set of traces of a specification.

However, this approach has a problem, that has lead to the development of more complicated notions of conformance for LTSes and IOTSes in particular. An, to avoid any doubt, this trace inclusion relation is *not* the **ioco** relation.

The problem is that traces record only positive aspects of behaviour: what *can* be done. What is missing is what *cannot* be done.

Consider Figure 5.1, where "?a" denotes the reception of 'a', and "!b" denotes the output of 'b'. The two systems have the same set of traces, namely

$$\epsilon, a, aa, aaa, \ldots, b, ba, baa, \ldots, ab, aba, abaa, \ldots, \ldots \tag{5.6}$$

However, there is a reason to believe that these systems should not be considered equivalent, because the system on the right is capable of "skipping" the output of 'b'. Intuitively, if one 'a' is sent to these systems, the leftmost system will reside in the topmost state, but the rightmost system could move to the bottom state. After the input of 'a', the system on the left could be excepted to produce 'b' if it would be delivered no further input. However, the system on the right could never produce 'b' if it would have moved to the state at the bottom.

This problem is caused by the lack of any notion of time in the ioco theory. For example, if there would be a timing rule that any output must be produced within one second since it has become possible, the rightmost system would have a trace "receive a, be silent for two seconds" that would not be possible for the leftmost system. But because time is not handled, it is impossible to observe from a set of traces whether outputs have been enabled or not in general.

The solution in the ioco theory is to make the lack of enabled outputs an observable situation in itself. The original authors explain this construct in different terms, but we use an alternative presentation that seems to be more compact.

5.2.3 Repetitive Quiescence

Suppose $\mathcal{I}$ is an IOTS, and let δ be an object outside the set L_τ. Denote by $\Delta(\mathcal{I})$ a new IOTS that has been obtained by $\mathcal{I}$ by the following transformation: for every state s such that

$$\neg\exists\alpha \in L_{\text{out}} : \exists s' : s \overset{\alpha}{\Longrightarrow} s', \tag{5.7}$$

add a self-loop $s \xrightarrow{\delta} s$. Finally, let L^δ_{out} denote the set $L_{\text{out}} \cup \{\delta\}$.

The operation Δ transforms an IOTS to another one such that the transformed IOTS can output δ's exactly whenever the original IOTS has no outputs enabled, i.e. can only wait for input.

We then take L^δ_{out} as the new set of output messages, and consider all testers over L_{in} and L^δ_{out}, and define

$$\mathsf{obs}(\mathscr{T}, \mathscr{I}) = \text{traces}(\mathscr{T} \,||\, \Delta(\mathscr{I})). \tag{5.8}$$

In other words, we take all traces of the resulting composed system, but now so that the traces include statements by the SUT or the specification that no outputs are currently enabled (which the tester must accept). This makes the lack of output an observable thing. The name for this construct in ioco theory is "repetitive quiescence". Quiescence means lack of output and is marked by the pseudo-output δ. Repetitive refers to the fact that many δ's can appear on a trace. The traces with δ's are called suspension traces.

On the bottom line, the ioco conformance relation for our purposes is then given by

$$\mathscr{X} \textbf{ ioco } \mathscr{S} \iff \text{traces}(\Delta(\mathscr{X})) \subseteq \text{traces}(\Delta(\mathscr{S})) \tag{5.9}$$

where $\mathscr{X}$ is an implementation, $\mathscr{S}$ a specification. The relation is to be read "$\mathscr{X}$ conforms to $\mathscr{S}$ in the sense of **ioco**". The reasoning here is equivalent to that in the previous section.

The original definition is slightly more complex, explaining the behaviour in those cases where the specification $\mathscr{S}$ is not an IOTS. Non-IOTS specifications are known as "partial specifications" in the ioco theory, but they can be dispensed with without any loss of generality.

Exercises

Exercise 5.1. Let C be a function from traces to natural numbers that countes the number of occurrences of δ. For example, $C(aba\delta\delta b) = 2$.

Let then obs_n be defined as

$$\mathsf{obs}_n(\mathscr{T}, \mathscr{I}) = \{T \mid T \in \text{traces}(\mathscr{T} \,||\, \Delta(\mathscr{I})) \wedge C(T) \leq n\}. \tag{5.10}$$

That is, obs_n corresponds to the capability of observing n quiescences (δ's) but no more.

Show that for any n, there exist a tester $\mathscr{T}$, and two IOTSes $\mathscr{X}$ and $\mathscr{S}$ such that

$$\mathsf{obs}_n(\mathscr{T}, \mathscr{X}) \subseteq \mathsf{obs}_n(\mathscr{T}, \mathscr{S}) \tag{5.11}$$

but that

$$\mathsf{obs}_{n+1}(\mathscr{T}, \mathscr{X}) \not\subseteq \mathsf{obs}_{n+1}(\mathscr{T}, \mathscr{S}). \tag{5.12}$$

(In the context of ioco theory, the special case with $n = 1$ corresponds to *quiescent traces* and the **conf** conformance relation. The special case with $n = 0$ is the *trace preorder*, which is exactly the plain trace set inclusion relation.)

Exercise 5.2. In this exercise we consider the lack of time in the ioco theory.

Suppose that we "execute" IOTSes like this: if we are in a state where at least one input is available, but there are also internal actions or output actions possible, we wait for one second for an input, and if no input arrives, we take an internal action or an output action. Otherwise, if only inputs are possible, we wait indefinitely for input. If only internal and output actions are possible, we take immediately one. Assume IOTSes are finite and that internal choices are taken randomly with flat probability distributions.

Explain informally why in this case the special δ-action is not needed in practice.

Chapter 6

Solutions to Exercises

2.1 (p. 22)

We can first sort the traces in the order of the end time stamp, because having a lesser time stamp is a prerequisite for being a prefix. We get:

$$T_5 = \langle \emptyset, 0 \rangle \tag{6.1}$$
$$T_6 = \langle \emptyset, 1 \rangle \tag{6.2}$$
$$T_2 = \langle \{\langle A, 1\rangle, \langle B, 1.5\rangle\}, 2.5 \rangle \tag{6.3}$$
$$T_1 = \langle \{\langle A, 1\rangle, \langle B, 2\rangle\}, 3 \rangle \tag{6.4}$$
$$T_3 = \langle \{\langle A, 1\rangle, \langle B, 2\rangle, \langle C, 3.5\rangle\}, 4 \rangle \tag{6.5}$$
$$T_4 = \langle \{\langle A, 1\rangle, \langle B, 2\rangle\}, 4 \rangle \tag{6.6}$$

Now T_5 is the empty trace which we know is the prefix of any other trace. T_6 is also a prefix of all the traces below it. Note that T_6 represents the fact that no event occurred *before* $t = 1$; it does not say anything about what happens *at* $t = 1$. Trace T_2 is not a prefix of any of the remaining traces. T_1 is a prefix of T_3, because the C event in T_3 occurs after the end time stamp of T_1. It is also a prefix of T_4. T_3 is not a prefix of T_4 because the time stamps are the same, and, besides, there is a difference in the event sets.

The solution is thus

$$\langle 1,3\rangle, \langle 1,4\rangle, \langle 5,1\rangle, \langle 5,2\rangle, \langle 5,3\rangle, \langle 5,4\rangle, \langle 5,6\rangle, \langle 6,1\rangle, \langle 6,2\rangle, \langle 6,3\rangle, \langle 6,4\rangle. \tag{6.7}$$

□

2.2 (p. 22)

(Reflexivity) Let $T = \langle E, t\rangle$. By definition, T is a prefix of T if and only if $t \leq t$ and $E = \{\langle \alpha, \tilde{t}\rangle \mid \langle \alpha, \tilde{t}\rangle \in E \wedge \tilde{t} < t\}$. By definition of trace, for any $\langle \alpha, \tilde{t}\rangle \in E$, $\tilde{t} < t$ holds. Therefore the equality for E holds, and $t \preceq t$.

(Transitivity) Let $T_i = \langle E_i, t_i\rangle$ for $i \in \{1,2,3\}$. Suppose $T_1 \preceq T_2$ and $T_2 \preceq T_3$. Then $t_1 \leq t_2 \leq t_3$, hence $t_1 \leq t_3$. It remains to show

$$E_1 = \{\langle \alpha, \tilde{t}\rangle \mid \langle \alpha, \tilde{t}\rangle \in E_3 \wedge \tilde{t} < t_1\}. \tag{6.8}$$

But now

$$\langle \alpha, \tilde{t}\rangle \in E_1 \iff \langle \alpha, \tilde{t}\rangle \in E_2 \wedge \tilde{t} < t_1 \tag{6.9}$$

and

$$\langle \alpha, \tilde{t}\rangle \in E_2 \iff \langle \alpha, \tilde{t}\rangle \in E_3 \wedge \tilde{t} < t_2 \tag{6.10}$$

by assumption. Hence

$$\langle \alpha, \tilde{t}\rangle \in E_1 \iff \langle \alpha, \tilde{t}\rangle \in E_3 \wedge \tilde{t} < t_1, t_2 \tag{6.11}$$

The result follows as $t_1 \leq t_2$.

(Antisymmetricity) Suppose $\langle E, t\rangle \prec \langle E', t'\rangle$. Hence $t < t'$. Therefore $\neg t' \leq t$, thus $\langle E', t'\rangle \not\preceq \langle E, t\rangle$. Hence $\preceq$ is antisymmetric. □

2.3 (p. 22)

Easily,

$$\mathbf{Pfx}(T) \subseteq \{T^* \mid T^* \preceq T\} \subseteq \{T^* \mid T^* \preceq T'\} \subseteq \mathbf{Pfx}(T)'. \tag{6.12}$$

□

2.4 (p. 22)

Case 1: ERROR. The specification does not allow for any I/O whatsoever.

Case 2: FAIL. The output message B is invalid.

Case 3: ERROR. The input message A is invalid.

Case 4: PASS. There is no I/O until time 0.5, which is correct.

2.5 (p. 22) For a trace T, let us denote by $T \triangleright K$ the set

$$T \triangleright K = \{\langle E, K\rangle \mid T \preceq \langle E, K\rangle \wedge \langle E, K\rangle \in \mathbf{Tr}(S)\}. \tag{6.13}$$

In other words, this is the set of those extensions of T in $\mathbf{Tr}(S)$ whose end time stamp is K. By the extendibility requirement, for every T_i, it holds that $T_i \triangleright K \neq \emptyset$.

Now note that $T \in T_i \triangleright K$ implies that $T \in T_j \triangleright K$ for all $j < i$, because $T_j \preceq T_i$. Hence, for $j \leq i$, $T_i \triangleright K \subseteq T_j \triangleright K$.

Therefore, let

$$\mathcal{I} = \bigcap\{T_i \triangleright K \mid i \geq 1\}. \tag{6.14}$$

Assume $\mathcal{I} = \emptyset$. Because of the inclusion chain $T_1 \triangleright K \supseteq T_2 \triangleright K \supseteq \cdots$, it must be that there exists i such that $T_i \triangleright K = \emptyset$. But this is not possible, as we have argued. Hence, the intersection $\mathcal{I}$ is not empty. By definition, every element of $\mathcal{I}$ belongs to $\mathbf{Tr}(S)$. Let $T^* = \langle E^*, K\rangle$ be one arbitrarily chosen element of $\mathcal{I}$.

By assumption, for every N there exists i such that $|E_i| > N$. Now note that $E_i \subseteq E^*$. Hence also $|E^*| > N$. Because this holds for an arbitrary N, it must be that $|E^*|$ is greater than any natural number. Hence, E^* is infinite. But, by definition, every trace has a finite number of events. Thus, T^* is not a trace, yet it belongs to the set of traces $\mathbf{Tr}(S)$. This is a contradiction. Hence the original claim. □

2.6 (p. 22) Suppose $T = \langle E, t\rangle$, S is a specification, $T \notin \mathbf{Tr}(S)$. Suppose there exists $\delta \in K$ such that $D_\delta \subseteq \Sigma_{\text{in}}$ and, for the sake of contradiction, that there exists also $\gamma \in K$ such that $D_\gamma \subseteq \Sigma_{\text{out}}$. Without loss of generality we assume $\delta \leq \gamma$ and proceed to derive a contradiction.

Because $\delta \leq \gamma$, we have $X_\delta \supseteq X_\gamma$. Hence it holds also that $D_\delta \supseteq D_\gamma$. Therefore, because Σ_{in} and Σ_{out} are distinct, it must hold that $D_\gamma = \emptyset$ (otherwise $D_\delta \not\subseteq \Sigma_{\text{in}}$).

X_γ is nonempty because S is a specification (every valid trace has at least one arbitrarily long extension). But every trace of X_γ differs from T at least at one time stamp. Therefore D_γ cannot be empty. Thus the final contradiction has been reached, and the proof is complete. □

3.1 (p. 30) The trace is valid. □

3.2 (p. 30) The trace is valid. □

3.3 (p. 30) The trace is invalid, and the verdict is FAIL. After 0.5 seconds the system moves on to silently discard any further inputs, and the output 2 is never produced. □

3.4 (p. 30) The trace is invalid. The verdict is ERROR, because any execution that begins with input 1 is bound to crash at the `require` procedure call. □

3.5 (p. 30) The trace is valid. □

3.6 (p. 31) The trace is invalid, but the verdict is CONF. The only valid execution is to output 2 at $t = 1$, but at this time there is input in the trace. □

5.1 (p. 40) Consider Figure 6.1. For any value of n, let $\mathscr{T}$ be the tester in this figure, $\mathscr{X}$ the implementation, and $\mathscr{S}$ the specification. Every trace that with this tester can reach the rightmost state of either the implementation or the specification must contain at least n δ's (this is easy to verify, as there must be as many δ's as there are a-inputs before the rightmost states). Therefore, when "execution" has reached the rightmost states, no more quiescences (δ's) can be counted in, and the δ-free trace continuations are equivalent for the implementation and the specification.

On the other hand, it is possible to reach the rightmost states with *exactly* n δ's. Hence, if it is possible to count one more quiescence, it is possible to observe that the implementation has taken the backwards-going arc, which results in quiescence, which is not possible for the specification. □

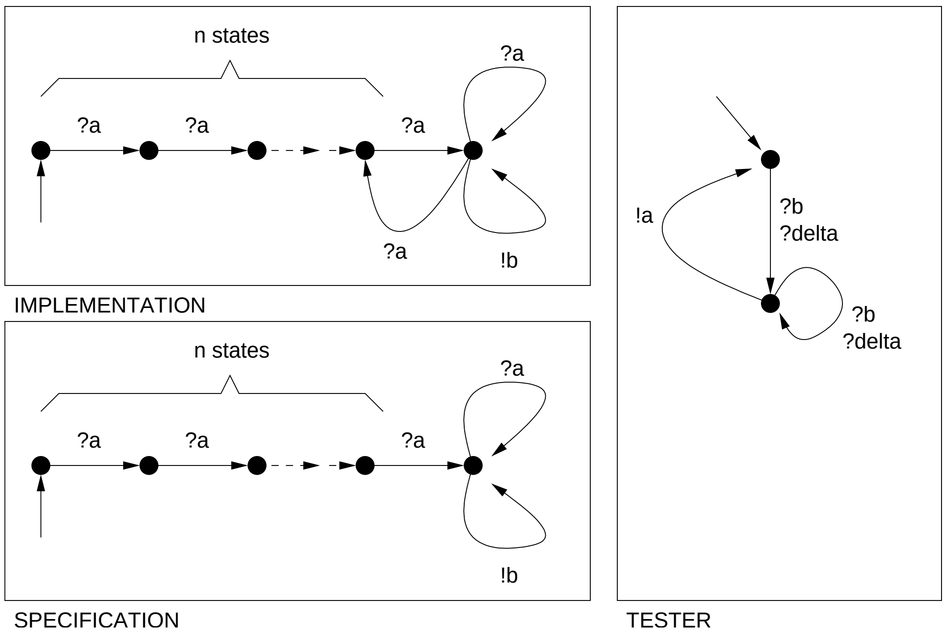

Figure 6.1: An implementation, a specification and a tester.

5.2 (p. 40)

In general, the δ-action is needed to be able to detect cases where no outputs will be made (in the future) before further input has been first received.

Under this timed interpretation, however, things change. If a tester just waits, an implementation performs random walk with speed of at least one transition in second, until it ends in a state with only input actions possible. Now a random walk in the IOTS eventually leads to a strongly connected component, and this strongly connected component is either quiescent or not. If it is not quiescent, then the probability of receiving an output approaches one when time passes. If it is quiescent, the probability becomes zero. By waiting long enough, it is possible to determine with arbitrarily small probability for error whether the system is quiescent or not. Certainly, for practical systems, this would be even easier.

The main point is that within a model like this, a system that is not quiescent will respond in a finite excepted time with an output. Therefore, non-quiescences and quiescences are both (at least probabilistically) observable. In practice, an abstraction for timeout (= quiescence) can be potentially useful, but adding time makes it no longer mandatory.

$\square$