Test Generation with Inputs, Outputs and Repetitive Quiescence

Jan Tretmans
Tele-Informatics and Open Systems Group
Department of Computer Science
University of Twente
P.O. Box 217, NL-7500 AE Enschede
tretmans@cs.utwente.nl

Abstract

This paper studies testing based on labelled transition systems, using the assumption that implementations communicate with their environment via inputs and outputs. Such implementations are formalized by restricting the class of transition systems to those systems that can always accept input actions, as in Input/Output Automata. Implementation relations, formalizing the notion of correctness of these implementations with respect to labelled transition system specifications, are defined analogous to the theories of testing equivalence and preorder, and refusal testing. A test generation algorithm is given which is proved to produce a sound and exhaustive test suite from a specification, i.e., a test suite that fully characterizes the set of correct implementations.

1 Introduction

Testing is an operational way to check the correctness of a system implementation by means of experimenting with it. Tests are applied to the implementation under test, and, based on observations made during the execution of the tests, a verdict about the correct functioning of the implementation is given. The correctness criterion that is to be tested is given in the system specification, preferably in some formal language. The specification is the basis for the derivation of test cases – when possible automatic derivation using a test generation algorithm.

Testing and verification are complementary techniques for analysis and checking of correctness of systems. While verification aims at proving properties about systems by formal manipulation on a mathematical model of the system, testing is performed by exercising the real, executing implementation (or an executable simulation model). Verification can give certainty about satisfaction of a required property, but this certainty only applies to the model of the system: any verification is only as good as the validity of the system model. Testing, being based on observing only a small subset of all possible instances of system behaviour, can never be complete: testing can only show the presence of errors, not their absence. But since testing can be applied to the real implementation, it is useful in those cases when a valid and reliable model of the system is difficult to build due to complexity, when the complete system is a combination of formal parts and parts which cannot be formally modelled (e.g., physical devices), when the model is proprietary (e.g., third party testing), or when the validity of a constructed model is to be checked with respect to the physical implementation.

Many different aspects of a system can be tested: Does the system do what it should do; i.e., does its behaviour comply with its functional specification (conformance testing)? How fast can the system perform its tasks (performance testing)? How does the system react if its environment does not behave as expected (robustness testing)? How long can we rely on the correct functioning of the system (reliability testing)? This paper focuses on conformance testing based on formal specifications; in particular, it aims at giving an algorithm for the generation of conformance test cases from transition system-based specifications.

The ingredients for defining such an algorithm comprise, apart from a formal specification, a class of implementations. An implementation under test, however, is a physical, real object that is in principle not amenable to formal reasoning. It is treated as a black box exhibiting behaviour and interacting with its environment. We can only deal with implementations in a formal way, if we make the assumption that any real implementation has a formal model with which we could reason formally. This formal model is only assumed to exist, but it is not known a priori. This assumption is referred to as the *test hypothesis* [Ber91, Tre92, ISO96]. Thus the test hypothesis allows to reason about implementations as if they were formal objects, and, consequently, to express the correctness of implementations with respect to specifications by a formal relation between such models of implementations and specifications. Such a relation is called an *implementation relation* [BAL$^+$90, ISO96]. Conformance testing now consists of performing experiments to decide whether the unknown model of the implementation relates to the specification according to the implementation relation. The experiments are specified in test cases. Given a specification, a test generation algorithm must produce a set of such test cases (a test suite). The test suite must be sound, i.e., it must give a negative verdict only if the implementation is incorrect and, if the implementation is incorrect, it must have a high probability to give a negative verdict.

One of the formalisms studied in the realm of conformance testing is that of *labelled transition systems*. A labelled transition system is a structure consisting of states with transitions, labelled with actions, between them. The formalism of labelled transition systems can be used for modelling the behaviour of processes, such as specifications, implementations and tests, and it serves as a semantic model for various formal languages, e.g., ACP [BK85], CCS [Mil89], and CSP [Hoa85]. Also (most parts of) the semantics of standardized languages like LOTOS [ISO89b], SDL [CCI92], and Estelle [ISO89a] can be expressed in labelled transition systems.

Traditionally, for labelled transition systems the term testing theory does not refer to conformance testing. Instead of starting with a specification to find a test suite to characterize the class of its conforming implementations, these testing theories aim at defining implementation relations, given a class of tests: a transition system p is equivalent to a system q if any test case in the class leads to the same observations with p as with q (or more generally, p relates to q if for all possible tests, the observations made of p are related in some sense to the observations made of q). Such a definition of an implementation relation by explicit use of the tests and observations that can discern them is referred to as an extensional definition. Many different relations can be defined by variations of the class of tests, the way they are executed, and the required relation between observations [DNH84, Abr87, DN87, Phi87, Gla90, Gla93].

Once an implementation relation has been defined, conformance testing involves finding a set of tests for one particular specification, that is in some sense minimal, and that can discriminate between correct and erroneous implementations of that specification. Conformance testing for labelled transition systems has been studied especially in the context of testing communication protocols with the language LOTOS, e.g., [BSS87, Bri88, PF90, Wez90, Led92, Tre92]. This paper uses both kinds of testing theories: first an implementation relation is defined extensionally, and then test generation from specifications for this particular relation is investigated.

Almost all of the testing theory for labelled transition systems mentioned above is based on synchronous, symmetric communication between processes: communication between two processes occurs if both processes offer to interact on a particular action and, if the interaction takes place it occurs synchronously in both participating processes. Both processes can propose and block the occurrence of an interaction; there is no distinction between input and output actions. For testing, a particular case where such communication occurs is the modelling of the interaction between a tester and an implementation under test during the execution of a test. We will refer to above theories as testing with symmetric interactions.

This paper approaches communication in a different manner by distinguishing explicitly between the inputs and the outputs of a system. Such a distinction is made, for example, in Input/Output Automata [LT89], Input-Output State Machines [Pha94], and Queue Contexts [TV92]. Outputs are actions that are initiated by and under control of the system, while input actions are initiated by and under control of the system's environment. A system can never refuse to perform its input actions, while its output actions cannot be blocked by the environment. Communication takes place between inputs of the system and outputs of the environment, or the other way around. It implies that an interaction is not symmetric anymore with respect to the communicating processes. Many real-life implementations allow such a classification of their actions in inputs and outputs, so it can be argued that such models have a closer link to reality. On the other hand, the input-output paradigm lacks some of the possibilities for abstraction, which can be a disadvantage when designing and specifying systems at a high level of abstraction. In an attempt to use the best of both worlds, this paper assumes that implementations communicate via inputs and outputs (as part of the test hypothesis), whereas specifications, although interpreting the same actions as inputs or outputs, are allowed to refuse their inputs, which implies that technically specifications are just transition systems.

The aim of this paper is to study implementation relations, conformance testing and test generation algorithms for labelled transition systems that communicate via inputs and outputs. The implementations are modelled by input-output transition systems, a special kind of labelled transition systems, where inputs are always enabled, and specifications are described as normal labelled transition systems. Input-output transition systems differ only marginally from the Input/Output Automata of [LT89]. These models are introduced in section 2. Implementation relations with inputs and outputs are defined extensionally following the ideas of testing equivalence and refusal testing [DNH84, DN87, Phi87, Lan90]. First, these existing relations, which are based on symmetric interactions, are recalled in section 3, and then their input-output versions are discussed in section 4. The first input-output relation, called input-output testing relation, is defined following a testing scenario à la [DNH84, DN87]. It is analogous to the scenario used in [Seg93] to obtain a testing characterization of the relation quiescent trace preorder on Input/Output Automata [Vaa91], and analogous results are obtained. The second relation, which is called input-output refusal relation, is defined with the testing scenario for refusal testing [Phi87, Lan90]. Weaker variants of both input-output relations are defined to allow for partial specifications. It will be shown that all defined relations can be simply and intuitively characterized in terms of only traces if a special action explicitly modelling the absence of outputs (repetitive quiescence, cf. [Vaa91]) is added. This special action has all the properties of, and can be considered as, a normal output action. The current paper generalizes [Seg93, Tre96], which considered only testing preorder with inputs and outputs, by also considering refusal testing and by showing that all relations can be expressed as special instances of a class of refusal-like implementation relations.

After having discussed the relevant implementation relations in section 4, section 5 starts formalizing conformance testing by introducing test cases, test suites, and how to run, execute and pass a test case. Finally, a test generation algorithm that produces provably correct test cases for any of the implementation relations of section 4 is developed in section 6. Analogous to the generalization of implementation relations, the algorithm of section 6 generalizes the one given in [Tre96] for refusal testing. Some concluding remarks and open problems are discussed in section 7. Elaborated proofs of theorems and propositions can be found in appendix A.

2 Models

The formalism of labelled transition systems is used as the basis for describing the behaviour of processes, such as specifications, implementations and tests.

Definition 2.1
A *labelled transition system* is a 4-tuple $\langle S, L, T, s_0 \rangle$ where

- S is a countable, non-empty set of *states*;
- L is a countable set of *labels*;
- $T \subseteq S \times (L \cup \{\tau\}) \times S$ is the *transition relation*;
- $s_0 \in S$ is the *initial state*.

□

The labels in L represent the observable actions of a system; the special label $\tau \notin L$ represents an unobservable, internal action. A transition $(s, \mu, s') \in T$ is denoted as $s \xrightarrow{\mu} s'$. A *computation* is a (finite) composition of transitions:

$$s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} s_2 \xrightarrow{\mu_3} \ \ldots \ \xrightarrow{\mu_{n-1}} s_{n-1} \xrightarrow{\mu_n} s_n$$

A *trace* captures the observable aspects of a computation; it is the sequence of observable actions of a computation. The set of all finite sequences of actions over L is denoted by L^*, with ϵ denoting the empty sequence. If $\sigma_1, \sigma_2 \in L^*$, then $\sigma_1 \cdot \sigma_2$ is the concatenation of σ_1 and σ_2.

We denote the class of all labelled transition systems over L by $\mathcal{LTS}(L)$. For technical reasons we restrict $\mathcal{LTS}(L)$ to labelled transition systems that are strongly convergent, i.e., ones that do not have infinite compositions of transitions with only internal actions. Some additional notations and properties are introduced in definitions 2.2 and 2.3.

Definition 2.2
Let $p = \langle S, L, T, s_0 \rangle$ be a labelled transition system with $s, s' \in S$, and let $\mu_{(i)} \in L \cup \{\tau\}$, $a_{(i)} \in L$, and $\sigma \in L^*$.

$$\begin{array}{lll}
s \xrightarrow{\mu_1 \cdot \ldots \cdot \mu_n} s' & =_{\text{def}} & \exists s_0, \ldots, s_n : \ s = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_n} s_n = s' \\
s \xrightarrow{\mu_1 \cdot \ldots \cdot \mu_n} & =_{\text{def}} & \exists s' : \ s \xrightarrow{\mu_1 \cdot \ldots \cdot \mu_n} s' \\
s \not\xrightarrow{\mu_1 \cdot \ldots \cdot \mu_n} & =_{\text{def}} & \text{not } \exists s' : \ s \xrightarrow{\mu_1 \cdot \ldots \cdot \mu_n} s' \\
s \xRightarrow{\epsilon} s' & =_{\text{def}} & s = s' \ \text{ or } \ s \xrightarrow{\tau \cdot \ldots \cdot \tau} s' \\
s \xRightarrow{a} s' & =_{\text{def}} & \exists s_1, s_2 : \ s \xRightarrow{\epsilon} s_1 \xrightarrow{a} s_2 \xRightarrow{\epsilon} s' \\
s \xRightarrow{a_1 \cdot \ldots \cdot a_n} s' & =_{\text{def}} & \exists s_0 \ldots s_n : \ s = s_0 \xRightarrow{a_1} s_1 \xRightarrow{a_2} \ldots \xRightarrow{a_n} s_n = s' \\
s \xRightarrow{\sigma} & =_{\text{def}} & \exists s' : \ s \xRightarrow{\sigma} s' \\
s \not\xRightarrow{\sigma} & =_{\text{def}} & \text{not } \exists s' : s \xRightarrow{\sigma} s'
\end{array}$$

□

We will not always distinguish between a transition system and its initial state: if $p = \langle S, L, T, s_0 \rangle$, we will identify the process p with its initial state s_0, e.g., we write $p \xRightarrow{\sigma}$ instead of $s_0 \xRightarrow{\sigma}$.

Definition 2.3

1. $\mathit{init}(p) \ =_{\text{def}} \ \{ \mu \in L \cup \{\tau\} \mid p \xrightarrow{\mu} \ \}$
2. $\mathit{traces}(p) \ =_{\text{def}} \ \{ \sigma \in L^* \mid p \xRightarrow{\sigma} \ \}$
3. $p \ \textbf{after} \ \sigma \ =_{\text{def}} \ \{ p' \mid p \xRightarrow{\sigma} p' \}$
4. p has *finite behaviour* if there is a natural number n such that all traces in *traces*(p) have length smaller than n.

5. p is *deterministic* if, for all $\sigma \in L^*$, $p \,\mathbf{after}\, \sigma$ has at most one element. If $\sigma \in \mathit{traces}(p)$, then $p \,\mathbf{after}\, \sigma$ is overloaded to denote this element.

$\square$

We represent a labelled transition system in the standard way, either by a tree or a graph, where nodes represent states and edges represent transitions (e.g., figure 1), or by a process-algebraic behaviour expression, with a syntax inspired by LOTOS [ISO89b]:

$$B \;=_{\mathrm{def}}\; \mathbf{stop} \;\mid\; a\,;B \;\mid\; \mathbf{i}\,;B \;\mid\; B \,\square\, B \;\mid\; B \,\|\, B \;\mid\; \Sigma\, \mathcal{B} \;\mid\; P$$

Here $a \in L$, $\mathcal{B}$ is a countable set of behaviour expressions, and $P \in \mathcal{P}$ is a process variable. The operational semantics of a behaviour expression with respect to an environment $\{P := B_P | P \in \mathcal{P}\}$ of process definitions is given in the standard way by the following axioms and inference rules, which define for each behaviour expression, in finitely many steps, all its possible transitions (**stop** has no transitions, and note that not every behaviour expression represents a transition system in $\mathcal{LTS}(L)$, e.g., the transition system defined by $P := \mathbf{i}; P$ is not strongly convergent):

$$\begin{array}{lll}
 & \vdash & a; B \xrightarrow{a} B \\
 & \vdash & \mathbf{i}; B \xrightarrow{\tau} B \\
B_1 \xrightarrow{\mu} B_1',\ \mu \in L \cup \{\tau\} & \vdash & B_1 \square B_2 \xrightarrow{\mu} B_1' \\
B_2 \xrightarrow{\mu} B_2',\ \mu \in L \cup \{\tau\} & \vdash & B_1 \square B_2 \xrightarrow{\mu} B_2' \\
B_1 \xrightarrow{\tau} B_1' & \vdash & B_1 \,\|\, B_2 \xrightarrow{\tau} B_1' \,\|\, B_2 \\
B_2 \xrightarrow{\tau} B_2' & \vdash & B_1 \,\|\, B_2 \xrightarrow{\tau} B_1 \,\|\, B_2' \\
B_1 \xrightarrow{a} B_1',\ B_2 \xrightarrow{a} B_2',\ a \in L & \vdash & B_1 \,\|\, B_2 \xrightarrow{a} B_1' \,\|\, B_2' \\
B \xrightarrow{\mu} B',\ B \in \mathcal{B},\ \mu \in L \cup \{\tau\} & \vdash & \Sigma\, \mathcal{B} \xrightarrow{\mu} B' \\
B_P \xrightarrow{\mu} B',\ P := B_P,\ \mu \in L \cup \{\tau\} & \vdash & P \xrightarrow{\mu} B'
\end{array}$$

Communication between a process and its environment, both modelled as labelled transition systems, is based on symmetric interaction, as expressed by the composition operator $\|$. An interaction can occur if both the process and its environment are able to perform that interaction, implying that they can also both block the occurrence of an interaction. If both offer more than one interaction then it is assumed that by some mysterious negotiation mechanism they will agree on a common interaction. There is no notion of input or output, nor of initiative or direction. All actions are treated in the same way for both communicating partners.

Many real systems, however, communicate in a different manner. They do make a distinction between inputs and outputs, and one can clearly distinguish whether the initiative for a particular interaction is with the system or with its environment. There is a direction in the flow of information from the initiating communicating process to the other. The initiating process determines which interaction will take place. Even if the other one decides not to accept the interaction, this is usually implemented by first accepting it, and then initiating a new interaction in the opposite direction explicitly signalling the non-acceptance. One could say that the mysterious negotiation mechanism is made explicit by exchanging two messages: one to propose an interaction and a next one to inform the initiating process about the (non-)acceptance of the proposed interaction.

We use *input-output transition systems*, analogous to Input/Output Automata [LT89], to model systems for which the set of actions can be partitioned into *output actions*, for which the initiative to perform them is with the system, and *input actions*, for which the initiative is with the environment. If an input action is initiated by the environment, the system is always prepared to participate in such an interaction: all the inputs of a system are always enabled; they can never be refused. Naturally an input action of the system can only interact with an output of the environment, and vice versa, implying that output actions can never be blocked by the environment. Although the initiative for any interaction is in exactly one of the communicating processes, the communication is still synchronous: if an interaction occurs, it occurs at exactly the same time in both processes. The communication, however, is not symmetric: the communicating processes have different roles in an interaction.

Definition 2.4
An *input-output transition system* p is a labelled transition system in which the set of actions L is partitioned into input actions L_I and output actions L_U ($L_I \cup L_U = L,\ \ L_I \cap L_U = \emptyset$), and for which all input actions are always enabled in any state:

$$\text{whenever} \quad p \stackrel{\sigma}{\Longrightarrow} p' \quad \text{then} \quad \forall a \in L_I : \ p' \stackrel{a}{\Longrightarrow}$$

The class of input-output transition systems with input actions in L_I and output actions in L_U is denoted by $\mathcal{IOTS}(L_I, L_U) \subseteq \mathcal{LTS}(L_I \cup L_U)$.

□

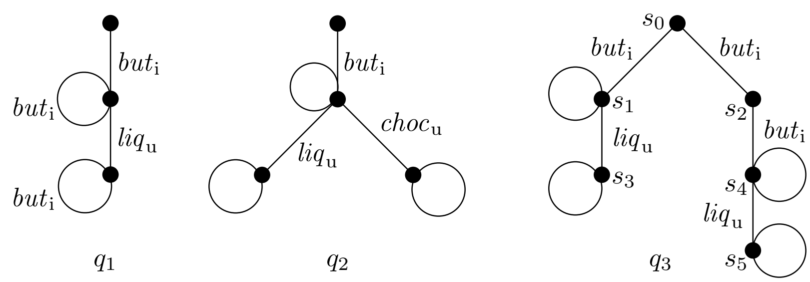

Figure 1: Input-output transition systems

Example 2.5
Figure 1 gives some input-output transition systems with $L_I = \{but_\mathrm{i}\}$ and $L_U = \{liq_\mathrm{u}, choc_\mathrm{u}\}$. In q_1 we can push the *button*, which is an input for the candy machine, and then the machine outputs *liquorice*. After the *button* has been pushed once, and also after the machine has released *liquorice*, any more pushing of the *button* has no effect: the machine makes a self-loop. In this paper we use the convention that a self-loop of a state that is not explicitly labelled is labelled with all inputs that cannot occur in that state (and also not via τ-transitions, cf. definition 2.4).

□

In the notational convention of input-output transition systems $a, b, c \ldots$ will denote input actions and $z, y, x, \ldots$ will denote output actions. Since input-output transition systems are labelled transition systems, all definitions for labelled transition systems apply. In particular, synchronous parallel communication can be expressed by $\|$, but now care should be taken that the outputs of one process interact with the inputs of the other.

Note that input-output transition systems differ marginally from Input/Output Automata [LT89]: instead of requiring *strong input enabling* as in [LT89] ($\forall a \in L_I : p' \stackrel{a}{\longrightarrow}$), input-output transition systems allow input enabling via internal transitions (*weak input enabling*, $\forall a \in L_I : p' \stackrel{a}{\Longrightarrow}$).

3 Implementation Relations with Symmetric Interactions

Before going to the test hypothesis that all implementations can be modelled by input-output transition systems in sections 4, 5, and 6, this section will briefly recall implementation relations and conformance testing based on the weaker hypothesis that implementations can be modelled as labelled transition systems. In this case correctness of an implementation with respect to a specification is expressed by an implementation relation $\mathbf{imp} \subseteq \mathcal{LTS}(L) \times \mathcal{LTS}(L)$. Many different relations have been studied in the literature, e.g., bisimulation equivalence [Mil89], failure equivalence and preorder [Hoa85], testing equivalence and preorder [DNH84, DN87], refusal testing [Phi87], and many others [Gla90, Gla93]. A straightforward example is *trace preorder* $\leq_{tr}$, which requires inclusion of sets of traces. The intuition behind this relation is that an implementation $i \in \mathcal{LTS}(L)$ may show only behaviour (in terms of traces of observable actions) which is specified in the specification $s \in \mathcal{LTS}(L)$.

Definition 3.1
Let $i, s \in \mathcal{LTS}(L)$, then $\quad i \leq_{tr} s \;\; =_{\text{def}} \;\; \mathit{traces}(i) \subseteq \mathit{traces}(s)$

□

Many implementation relations can be defined in an extensional way, which means that they are defined by explicitly comparing an implementation with a specification in terms of comparing the observations that an external observer can make [DNH84, DN87]. The intuition is that an implementation i correctly implements a specification s if any observation that can be made of i in any possible environment can be related to, or explained from, an observation of s in the same environment:

$$i \;\mathbf{imp}\; s \quad =_{\text{def}} \quad \forall u \in \mathcal{U} : \;\; \mathit{obs}(u, i) \;\; * \;\; \mathit{obs}(u, s) \tag{1}$$

By varying the class of external observers $\mathcal{U}$, the observations *obs* that an observer can make of i and s, and the relation $*$ between observations of i and s, many different implementation relations can be defined.

One of the relations that can be expressed following (1) is *testing preorder* $\leq_{te}$, which we formalize in a slightly different setting from the one in [DNH84, DN87]. It is obtained if labelled transition systems are chosen as observers $\mathcal{U}$, the relation between observations is set inclusion, and the observations are traces. These traces are obtained from computations of i or s, in parallel with an observer u, where a distinction is made between normal traces and completed traces, i.e., traces which correspond to a computation after which no more actions are possible.

Definition 3.2
Let $p, i, s \in \mathcal{LTS}(L)$, $\sigma \in L^*$, and $A \subseteq L$, then

1. $p \;\mathbf{after}\; \sigma \;\mathbf{refuses}\; A \quad =_{\text{def}} \;\; \exists p' : \; p \stackrel{\sigma}{\Longrightarrow} p' \;$ and $\; \forall a \in A : \; p' \stackrel{a}{\not\Longrightarrow}$
2. $p \;\mathbf{after}\; \sigma \;\mathbf{deadlocks} \quad =_{\text{def}} \quad p \;\mathbf{after}\; \sigma \;\mathbf{refuses}\; L$
3. The sets of *observations* obs_c and obs_t that an observer $u \in \mathcal{LTS}(L)$ can make of process $p \in \mathcal{LTS}(L)$ are given by the completed traces and the traces, respectively, of their synchronized parallel communication $u \,\|\, p$:

$$\begin{array}{lcl} \mathit{obs}_c(u,p) & =_{\text{def}} & \{\, \sigma \in L^* \mid \; (u \,\|\, p) \;\mathbf{after}\; \sigma \;\mathbf{deadlocks} \,\} \\ \mathit{obs}_t(u,p) & =_{\text{def}} & \{\, \sigma \in L^* \mid \; (u \,\|\, p) \stackrel{\sigma}{\Longrightarrow} \,\} \end{array}$$

4. $i \leq_{te} s \quad =_{\text{def}} \quad \forall u \in \mathcal{LTS}(L) : \quad \mathit{obs}_c(u,i) \subseteq \mathit{obs}_c(u,s) \;$ and $\; \mathit{obs}_t(u,i) \subseteq \mathit{obs}_t(u,s)$

□

The definitions in 3.2 are based on the occurrence, or absence, of observable actions. It is straightforward to show that on our class of strongly convergent transition systems these definitions correspond to those sometimes found in the literature, which also take internal actions into account:

$$p \;\mathbf{after}\; \sigma \;\mathbf{refuses}\; A \quad \text{iff} \quad \exists p' : \; p \stackrel{\sigma}{\Longrightarrow} p' \;\text{ and }\; \forall \mu \in A \cup \{\tau\} : \; p' \stackrel{\mu}{\not\longrightarrow} \tag{2}$$

The extensional definition of $\leq_{te}$ in definition 3.2 can be rewritten into an intensional characterization, i.e., a characterization in terms of properties of the labelled transition systems themselves. This characterization, given in terms of failure pairs, is known to coincide with failure preorder for strongly convergent transition systems [DN87, Tre92].

Proposition 3.3

$i \leq_{te} s \quad \text{iff} \quad \forall \sigma \in L^*,\ \forall A \subseteq L : \quad i \textbf{ after } \sigma \textbf{ refuses } A \ \text{ implies } \ s \textbf{ after } \sigma \textbf{ refuses } A$

□

A weaker implementation relation that is strongly related to $\leq_{te}$ is the relation **conf** [BSS87]. It is a modification of $\leq_{te}$ by restricting all observations to only those traces that are contained in the specification s. This restriction is in particular used in conformance testing. It makes testing a lot easier: only traces of the specification have to be considered, not the huge complement of this set, i.e., the traces not explicitly specified. In other words, **conf** requires that an implementation does what it should do, not that it does not do what it is not allowed to do. So a specification only partially prescribes the required behaviour of the implementation. Several test generation algorithms have been developed for the relation **conf** [Bri88, PF90, Wez90, Tre92].

Definition 3.4

$$\begin{array}{llll} i \textbf{ conf } s & =_{\text{def}} & \forall u \in \mathcal{LTS}(L) : & (obs_c(u,i) \cap traces(s)) \subseteq obs_c(u,s) \\ & & \quad\quad \text{and} & (obs_t(u,i) \cap traces(s)) \subseteq obs_t(u,s) \end{array}$$

□

Proposition 3.5

$i \textbf{ conf } s \quad \text{iff} \quad \forall \sigma \in traces(s),\ \forall A \subseteq L : \quad i \textbf{ after } \sigma \textbf{ refuses } A \ \text{ implies } \ s \textbf{ after } \sigma \textbf{ refuses } A$

□

A relation with more discriminating power than testing preorder is obtained, following (1), by having more powerful observers that can detect not only the occurrence of actions but also the absence of actions, i.e., refusals [Phi87]. We follow [Lan90] in modelling the observation of a refusal by adding a special label $\theta \notin L$ to observers: $\mathcal{U} = \mathcal{LTS}(L_\theta)$, where we write L_θ for $L \cup \{\theta\}$. While observing a process, a transition labelled with θ can only occur if no other transition is possible. In this way the observer knows that the process under observation cannot perform the other actions it offers. A parallel synchronization operator $\rceil|$ is introduced, which models the communication between an observer with θ-transitions and a normal process, i.e., a transition system without θ-transitions. The implementation relation defined in this way is called *refusal preorder* $\leq_{rf}$.

Definition 3.6

1. The operator $\rceil| : \mathcal{LTS}(L_\theta) \times \mathcal{LTS}(L) \to \mathcal{LTS}(L_\theta)$ is defined by the following inference rules:

$$\begin{array}{lll} u \xrightarrow{\tau} u' & \vdash & u\rceil|\,p \xrightarrow{\tau} u'\rceil|\,p \\ p \xrightarrow{\tau} p' & \vdash & u\rceil|\,p \xrightarrow{\tau} u\rceil|\,p' \\ u \xrightarrow{a} u',\ p \xrightarrow{a} p',\ a \in L & \vdash & u\rceil|\,p \xrightarrow{a} u'\rceil|\,p' \\ u \xrightarrow{\theta} u',\ u \xrightarrow{\tau}\!\!\!\!\!/\ ,\ p \xrightarrow{\tau}\!\!\!\!\!/\ ,\ \forall a \in L :\ u \xrightarrow{a}\!\!\!\!\!/ \ \text{ or } \ p \xrightarrow{a}\!\!\!\!\!/ & \vdash & u\rceil|\,p \xrightarrow{\theta} u'\rceil|\,p \end{array}$$

2. The sets of *observations* obs^θ_c and obs^θ_t that an observer $u \in \mathcal{LTS}(L_\theta)$ can make of process $p \in \mathcal{LTS}(L)$ are given by the completed traces and the traces, respectively, of the synchronized parallel communication $\rceil|$ of u and p:

$$\begin{array}{lll} obs^\theta_c(u,p) & =_{\text{def}} & \{\ \sigma \in L^*_\theta \ \mid\ (u\rceil|\,p) \textbf{ after } \sigma \textbf{ deadlocks }\ \} \\ obs^\theta_t(u,p) & =_{\text{def}} & \{\ \sigma \in L^*_\theta \ \mid\ (u\rceil|\,p) \overset{\sigma}{\Longrightarrow}\ \} \end{array}$$

3. $i \leq_{rf} s \quad =_{\text{def}} \quad \forall u \in \mathcal{LTS}(L_\theta) : \quad obs^\theta_c(u,i) \subseteq obs^\theta_c(u,s) \ \text{ and } \ obs^\theta_t(u,i) \subseteq obs^\theta_t(u,s)$

□

A corresponding intensional characterization of refusal preorder can be given in terms of failure traces [Gla90, Lan90]. A failure trace is a trace in which both actions and refusals, represented by sets of refused actions, occur. To express this, the transition relation $\longrightarrow$ is extended with refusal transitions: self-loop transitions labelled with a set of actions $A \subseteq L$, expressing that all actions in A can be refused. The transition relation $\Longrightarrow$ (definition 2.2) is then extended analogously to $\stackrel{\varphi}{\Longrightarrow}$ with $\varphi \in (L \cup \mathcal{P}(L))^*$.

Definition 3.7
Let $p \in \mathcal{LTS}(L)$ and $A \subseteq L$.

1. $p \xrightarrow{A} p' \quad =_{\text{def}} \quad p = p'$ and $\forall \mu \in A \cup \{\tau\} :\ p \stackrel{\mu}{\not\longrightarrow}$
2. The *failure traces* of p are: $Ftraces(p) \ =_{\text{def}} \ \{ \varphi \in (L \cup \mathcal{P}(L))^* \mid p \stackrel{\varphi}{\Longrightarrow} \}$

□

Proposition 3.8
$i \leq_{rf} s$ iff $Ftraces(i) \subseteq Ftraces(s)$

□

We conclude this section by relating the implementation relations based on symmetric interactions and illustrating them using the candy machines over $L = \{but, choc, liq, bang\}$ in figure 2. These examples also illustrate the inequalities of proposition 3.9.

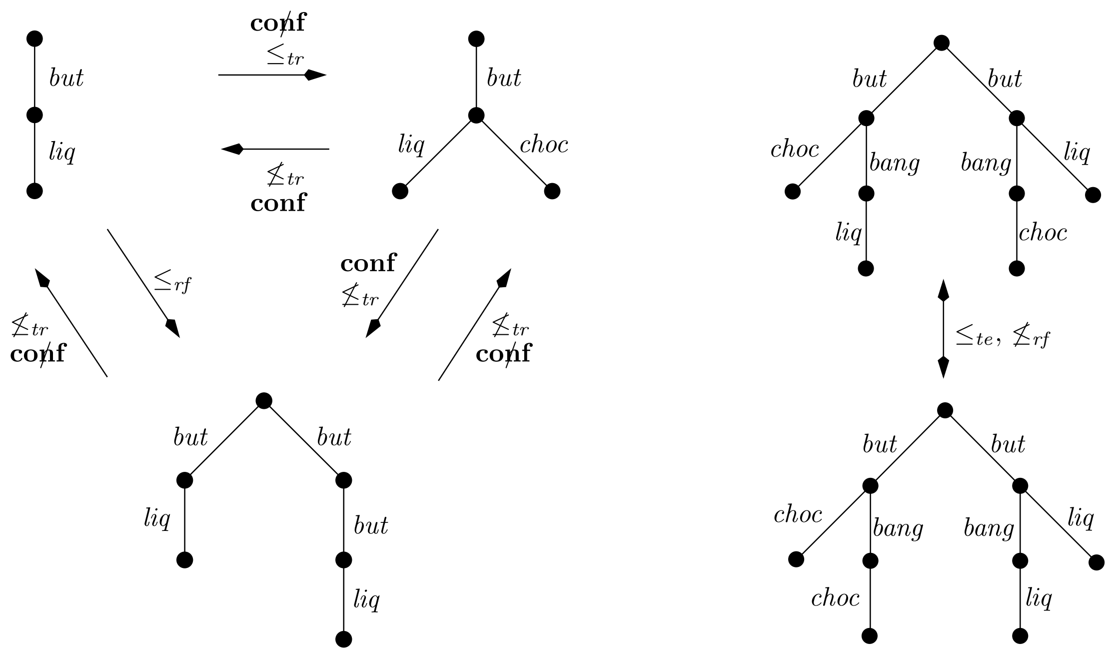

Figure 2: Implementation relations with symmetric interactions

Proposition 3.9

1. $\leq_{tr}$, $\leq_{te}$, $\leq_{rf}$ are preorders; **conf** is reflexive, but not transitive.
2. $\leq_{rf} \ \subset \ \leq_{te} \ = \ \leq_{tr} \cap \ \mathbf{conf}$

□

4 Implementation Relations with Inputs and Outputs

We now make the test assumption that implementations can be modelled by input-output transition systems: we consider implementation relations $\mathbf{imp} \subseteq \mathcal{IOTS}(L_I, L_U) \times \mathcal{LTS}(L_I \cup L_U)$. Like the relations based on symmetric interactions in section 3, we define them extensionally following (1).

4.1 Input-output testing relation

The implementation relations $\leq_{te}$ and **conf** were defined by relating the observations made of the implementation by a symmetrically interacting observer $u \in \mathcal{LTS}(L)$ to the observations made of the specification (definitions 3.2 and 3.4). An analogous testing scenario can be defined for input-output transition systems using the fact that communication takes place along the lines explained in section 2: the input actions of the observer synchronize with the output actions of the implementation, and vice versa, so an input-output implementation in $\mathcal{IOTS}(L_I, L_U)$ communicates with an 'output-input' observer in $\mathcal{IOTS}(L_U, L_I)$. In this way the *input-output testing relation* $\leq_{iot}$ is defined between $i \in \mathcal{IOTS}(L_I, L_U)$ and $s \in \mathcal{LTS}(L_I \cup L_U)$ by requiring that any possible observation made of i by any 'output-input' transition system is a possible observation of s by the same observer (cf. definition 3.2).

Definition 4.1
Let $i \in \mathcal{IOTS}(L_I, L_U)$, $s \in \mathcal{LTS}(L_I \cup L_U)$, then

$$i \leq_{iot} s \quad =_{\mathrm{def}} \quad \forall u \in \mathcal{IOTS}(L_U, L_I): \quad obs_c(u, i) \subseteq obs_c(u, s) \;\text{ and }\; obs_t(u, i) \subseteq obs_t(u, s)$$

□

Note that, despite what was said above about the communication between the implementation and the observer, the observations made of s are based on the communication between an input-output transition system and a standard labelled transition system, since s need not be an input-output system. Technically there is no problem in making such observations: the definitions of obs_c, obs_t, $\|$, and $.\,\mathbf{after}\,.\,\mathbf{deadlocks}$ apply to labelled transition systems, not only to input-output transition systems. Below we will elaborate on this possibility to have $s \notin \mathcal{IOTS}$.

In [Seg93] the testing scenario of testing preorder [DNH84, DN87] was applied to define a relation on Input/Output Automata completely analogous to definition 4.1. It was shown to yield the implementation relation *quiescent trace preorder* introduced in [Vaa91]. Although we are more liberal with respect to the specification, $s \in \mathcal{LTS}(L_I \cup L_U)$, and input-output transition systems differ marginally from Input/Output Automata, exactly the same intensional characterization is obtained: $\leq_{iot}$ is fully characterized by trace inclusion and inclusion of quiescent traces. A trace is *quiescent* if it may lead to a state from which the system cannot proceed autonomously, without inputs from its environment, i.e., a state from which no outputs or internal actions are possible.

Definition 4.2
Let $p \in \mathcal{LTS}(L_I \cup L_U)$.

1. A state s of p is *quiescent*, denoted by $\delta(s)$, if $\forall \mu \in L_U \cup \{\tau\}: \; s \xrightarrow{\mu}\!\!\!\!\!/\;\;$
2. A *quiescent trace* of p is a trace σ that may lead to a quiescent state: $\exists p' \in (\, p \,\mathbf{after}\, \sigma\,): \; \delta(p')$
3. The set of quiescent traces of p is denoted by $Qtraces(p)$.

□

Proposition 4.3
$i \leq_{iot} s$ iff $traces(i) \subseteq traces(s)$ and $Qtraces(i) \subseteq Qtraces(s)$

□

Sketch of the proof
Comparing with the analogous definition and proposition for $\leq_{te}$ (definition 3.2 and proposi-

tion 3.3), we see that the observation of deadlock of $u \| i$ can only occur if i is in a state where it cannot produce any output (a quiescent state) and u is in a state where it cannot produce any input (input with respect to i). It follows then that for inclusion of obs_c it suffices to consider only quiescent traces. Inclusion of obs_t corresponds to inclusion of traces.

□

Comparing the intensional characterization of $\leq_{iot}$ in proposition 4.3 with the one for $\leq_{te}$ (proposition 3.3), we see that the restriction to input-output systems simplifies the corresponding intensional characterization. Instead of sets of pairs consisting of a trace and a set of actions (failure pairs), it suffices to look at just two sets of traces.

Another characterization of $\leq_{iot}$ can be given by collecting for each trace all possible outputs that a process may produce after that trace, including a special action δ that indicates possible quiescence. The special label $\delta \notin L$ indicates the absence of output actions in a state, i.e., it makes the observation of absence of outputs (quiescence) into an explicitly observable event. Proposition 4.5 then states that an implementation is correct according to $\leq_{iot}$ if all outputs it can produce after any trace σ can also be produced by the specification. Since this also holds for δ, the implementation may show no outputs only if the specification can do so.

Definition 4.4
Let p be a state in a transition system, and let P be a set of states, then

1. $out(p) \;=_{\mathrm{def}}\; \{\, x \in L_U \mid p \xrightarrow{x} \,\} \cup \{\, \delta \mid \delta(p) \,\}$
2. $out(P) \;=_{\mathrm{def}}\; \bigcup \{\, out(p) \mid p \in P \,\}$

□

Proposition 4.5
$i \leq_{iot} s \quad \text{iff} \quad \forall \sigma \in L^* : \; out(\, i \;\mathbf{after}\; \sigma\,) \;\subseteq\; out(\, s \;\mathbf{after}\; \sigma\,)$

□

Sketch of the proof
Using the facts that $\sigma \in Qtraces(p)$ iff $\delta \in out(\, p \;\mathbf{after}\; \sigma\,)$ and $\sigma \in traces(p)$ iff $out(\, p \;\mathbf{after}\; \sigma\,) \neq \emptyset$, the proposition follows immediately from proposition 4.3.

□

Figure 3: Two non-input-output specifications

Example 4.6
Using proposition 4.5, it follows that $q_1 \leq_{iot} q_2$ (figure 1): an implementation capable of only producing *liquorice* conforms to a specification that prescribes to produce either *liquorice* or *chocolate*. Although q_2 looks deterministic, in fact it specifies that after *button* there is a nondeterministic choice between supplying *liquorice* or *chocolate*. It also implies that for this kind of testing q_2 is equivalent to $but_{\mathrm{i}}; liq_{\mathrm{u}}; \mathbf{stop} \;\Box\; but_{\mathrm{i}}; choc_{\mathrm{u}}; \mathbf{stop}$ (omitting the input self-loops), an equivalence which does not hold for $\leq_{te}$ in the symmetric case. If we want to specify a machine that produces both *liquorice* and *chocolate*, then two buttons are needed to select the respective candies:

$$liq\text{-}button \;;\; liq_{\mathrm{u}} \;;\; \mathbf{stop} \;\;\Box\;\; choc\text{-}button \;;\; choc_{\mathrm{u}} \;;\; \mathbf{stop}$$

On the other hand, $q_2 \not\leq_{iot} q_1, q_3$: if the specification prescribes to produce only *liquorice*, then an implementation should not have the possibility to produce *chocolate*: $choc_{\mathrm{u}} \in out(\, q_2 \, \mathbf{after}\, but_{\mathrm{i}}\,)$, while $choc_{\mathrm{u}} \notin out(\, q_1 \, \mathbf{after}\, but_{\mathrm{i}}\,)$, $choc_{\mathrm{u}} \notin out(\, q_3 \, \mathbf{after}\, but_{\mathrm{i}}\,)$. We have $q_1 \leq_{iot} q_3$, but $q_3 \not\leq_{iot} q_1, q_2$, since q_3 may refuse to produce anything after the *button* has been pushed once, while both q_1 and q_2 will always output something. Formally: $\delta \in out(\, q_3 \, \mathbf{after}\, but_{\mathrm{i}}\,)$, while $\delta \notin out(\, q_1 \, \mathbf{after}\, but_{\mathrm{i}}\,), out(\, q_2 \, \mathbf{after}\, but_{\mathrm{i}}\,)$.

Figure 3 presents two non-input-output transition system specifications, but none of q_1, q_2, q_3 correctly implements s_1 or s_2; the problem occurs with non-specified input traces of the specification: $out(\, q_1 \, \mathbf{after}\, but_{\mathrm{i}} \cdot but_{\mathrm{i}}\,)$, $out(\, q_2 \, \mathbf{after}\, but_{\mathrm{i}} \cdot but_{\mathrm{i}}\,)$, $out(\, q_3 \, \mathbf{after}\, but_{\mathrm{i}} \cdot but_{\mathrm{i}}\,) \neq \emptyset$, while $but_{\mathrm{i}} \cdot but_{\mathrm{i}} \notin traces(s_1)$, $traces(s_2)$, so $out(\, s_1 \, \mathbf{after}\, but_{\mathrm{i}} \cdot but_{\mathrm{i}}\,)$, $out(\, s_2 \, \mathbf{after}\, but_{\mathrm{i}} \cdot but_{\mathrm{i}}\,) = \emptyset$.

□

The relation $\leq_{iot}$ does not require the specification to be an input-output transition system: a specification may have states that can refuse input actions. Such a specification is interpreted as an incompletely specified input-output transition system, i.e., a transition system where a distinction is made between inputs and outputs, but where some inputs are not specified in some states. The intention of such specifications often is that the specifier does not care about the responses of an implementation to such non-specified inputs. If a candy machine is specified to deliver liquorice after a button is pushed as in s_1 in figure 3, then it is intentionally left open what an implementation may do after the button is pushed twice: perhaps ignoring it, supplying one of the candies, or responding with an error message. Intuitively, q_1 would conform to s_1; however, $q_1 \not\leq_{iot} s_1$, as was shown in example 4.6. The implementation freedom, intended with non-specified inputs, cannot be expressed with the relation $\leq_{iot}$. From proposition 4.5 the reason can be deduced: since any implementation can always perform any sequence of input actions, and since from definition 4.4 it is easily deduced that $out(\, p \, \mathbf{after}\, \sigma\,) \neq \emptyset$ iff $p \stackrel{\sigma}{\Longrightarrow}$, we have that an $\leq_{iot}$-implementable specification must at least be able to perform any sequence of input actions. So the specification must be an input-output transition system, too, otherwise no $\leq_{iot}$-implementation can exist.

For Input/Output Automata a solution to this problem is given in [DNS95], using the so-called demonic semantics for process expressions. In this semantics a transition to a demonic process Ω is added for each non-specified input. Since Ω exhibits any behaviour, the behaviour of the implementation is not prescribed after such a non-specified input. We choose another solution to allow for non-input-output transition system specifications to express implementation freedom for non-enabled inputs: we introduce a weaker implementation relation. The discussion above immediately suggests how such a relation can be defined: instead of requiring inclusion of *out*-sets for all traces in L^* (proposition 4.5), the weaker relation requires only inclusion of *out*-sets for traces that are explicitly specified in the specification. This relation is called *i/o-conformance* **ioconf**, and, analogously to **conf**, it allows partial specifications which only state requirements for traces explicitly specified in the specification (cf. the relation between $\leq_{te}$ and **conf**, definitions 3.2 and 3.4, and propositions 3.3 and 3.5).

Definition 4.7
Let $i \in \mathcal{IOTS}(L_I, L_U)$, $s \in \mathcal{LTS}(L_I \cup L_U)$, then

$$i \;\mathbf{ioconf}\; s \;=_{\mathrm{def}}\; \forall \sigma \in traces(s): \quad out(\, i \, \mathbf{after}\, \sigma\,) \;\subseteq\; out(\, s \, \mathbf{after}\, \sigma\,)$$

□

Example 4.8
Consider again figures 1 and 3. Indeed, we have q_1 **ioconf** s_1, whereas we had $q_1 \not\leq_{iot} s_1$. According to **ioconf**, s_1 specifies only that after one *button*, *liquorice* must be produced; with **ioconf**, s_1 does not care what happens if the *button* is pushed twice, as was the case with $\leq_{iot}$.

On the other hand, q_2 $\not\!\!\mathbf{ioconf}$ s_1, since q_2 can produce more than *liquorice* after the *button* has been pushed: $out(\, q_2 \, \mathbf{after}\, but_{\mathrm{i}}\,) = \{liq_{\mathrm{u}}, choc_{\mathrm{u}}\} \not\subseteq \{liq_{\mathrm{u}}\} = out(\, s_1 \, \mathbf{after}\, but_{\mathrm{i}}\,)$. Moreover,

q_1, q_2 **ioconf** s_2, but q_3 **ioc$\not$onf** s_1, s_2, since $\delta \in out(\, q_3\,\textbf{after}\, but_{\mathrm{i}}\,)$, while $\delta \notin out(\, s_1\,\textbf{after}\, but_{\mathrm{i}}\,)$, $out(\, s_2\,\textbf{after}\, but_{\mathrm{i}}\,)$.

□

4.2 Input-output refusal relation

We have seen implementation relations with symmetric interactions based on observers without and with θ-label, which resulted in the relations $\leq_{te}$ and $\leq_{rf}$, respectively, and we have seen an implementation relation with inputs and outputs based on observers without θ-label. Naturally, the next step is an implementation relation with inputs and outputs based on observers with the power of the θ-label. The resulting relation is called the *input-output refusal relation* $\leq_{ior}$.

Definition 4.9
Let $i \in \mathcal{IOTS}(L_I, L_U)$, $s \in \mathcal{LTS}(L_I \cup L_U)$, then

$$i \leq_{ior} s \;\; =_{\text{def}} \;\; \forall u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\}) : \quad obs_c^{\theta}(u,i) \subseteq obs_c^{\theta}(u,s) \;\text{ and }\; obs_t^{\theta}(u,i) \subseteq obs_t^{\theta}(u,s)$$

□

A quiescent trace was introduced as a trace ending in the absence of outputs. Taking the special action δ, which was used in *out*-sets to indicate the absence of outputs, a quiescent trace $\sigma \in L^*$ can be written as a δ-ending trace $\sigma\cdot\delta \in (L \cup \{\delta\})^*$. Here, the special action δ appears always as the last action in the trace. If this special action δ is now treated as a completely normal action which can occur at any place in a trace, we obtain traces with repetitive quiescence. For example, the trace $\delta\cdot a\cdot\delta\cdot b\cdot x$ would mean intuitively that initially no outputs can be observed, then after input action a there is again no output, and then after input b is performed the output x can be observed. We write L_δ for $L \cup \{\delta\}$, and we call traces over L_δ *suspension traces*. The implementation relation $\leq_{ior}$ turns out to be characterized by inclusion of these suspension traces (and hence it could also be called *repetitive quiescence relation*). Since quiescence corresponds to a refusal of L_U (definition 3.7), it follows that suspension traces are exactly the failure traces in which only L_U occurs as refusal set, i.e., failure traces restricted to $(L \cup \{L_U\})^*$, and where δ is written for the refusal L_U.

Definition 4.10
The *suspension traces* of process $p \in \mathcal{LTS}(L)$ are: $\;Straces(p) \;=_{\text{def}}\; Ftraces(p) \;\cap\; (L \cup \{L_U\})^*$. For L_U occurring in a suspension trace, we write δ, so that a suspension trace $\sigma \in L_\delta^*$.

□

Proposition 4.11
$i \leq_{ior} s \quad \text{iff} \quad Straces(i) \subseteq Straces(s)$

□

Sketch of the proof
Analogous to the proof of proposition 4.3, and comparing with the corresponding situation for $\leq_{rf}$ (definition 3.6 and proposition 3.8), a refusal can only be observed if i is in a state where it cannot produce any output (a quiescent state) and u is in a state where it cannot produce any input (input with respect to i). So the only refusals of i that can be observed are L_U. As opposed to proposition 4.3, the normal traces are included in the suspension traces, so they need not be mentioned separately in the proposition.

□

An intensional characterization of $\leq_{ior}$ in terms of *out*-sets, analogous to proposition 4.5, is easily given by generalizing the definition of **after** (definition 2.3) in a straightforward way to suspension traces.

Proposition 4.12
$i \leq_{ior} s \quad \text{iff} \quad \forall \sigma \in L_\delta^* : \quad out(\, i\,\textbf{after}\,\sigma\,) \;\subseteq\; out(\, s\,\textbf{after}\,\sigma\,)$

□

Again, completely analogous to the definitions of **conf** and of **ioconf**, an implementation relation, called **ioco**, is defined by restricting inclusion of *out*-sets to suspension traces of the specification.

Definition 4.13

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

□

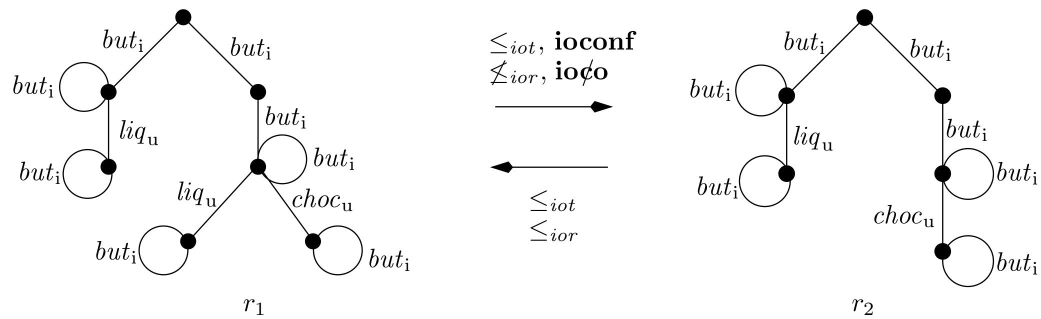

Figure 4: The difference between $\leq_{iot}$ and $\leq_{ior}$

Example 4.14

Examples 4.6 and 4.8 illustrated the implementation relations $\leq_{iot}$ and **ioconf**, respectively, using the processes in figures 1 and 3. Replacing $\leq_{iot}$ by $\leq_{ior}$ and **ioconf** by **ioco**, the same results are obtained for the processes in figures 1 and 3.

The difference between $\leq_{iot}$ and $\leq_{ior}$, and between **ioconf** and **ioco** is illustrated with the processes r_1 and r_2 in figure 4: $r_1 \leq_{iot} r_2$, but $r_1 \not\leq_{ior} r_2$; $\mathit{out}(\, r_1 \textbf{ after } but_{\mathrm{i}} \cdot \delta \cdot but_{\mathrm{i}}\,) = \{liq_{\mathrm{u}}, choc_{\mathrm{u}}\}$ and $\mathit{out}(\, r_2 \textbf{ after } but_{\mathrm{i}} \cdot \delta \cdot but_{\mathrm{i}}\,) = \{choc_{\mathrm{u}}\}$. Intuitively, after pushing the *button*, we observe that nothing is produced by the machine, so we push the *button* again. Machine r_1 may then produce either *liquorice* or *chocolate*, while machine r_2 will always produce *chocolate*. When we use the relation $\leq_{iot}$, the observation always terminates after observing that nothing is produced. Hence, there is no way to distinguish between entering the left or the right branch of r_1 or r_2; after the *button* is pushed twice, both machines may produce either *liquorice* or *chocolate*: $\mathit{out}(\, r_{1,2} \textbf{ after } but_{\mathrm{i}} \cdot but_{\mathrm{i}}\,) = \{liq_{\mathrm{u}}, choc_{\mathrm{u}}\}$.

□

4.3 Relating relations with inputs and outputs

Two kinds of observations were used in the extensional definitions of testing preorder $\leq_{te}$ (definition 3.2), refusal preorder $\leq_{rf}$ (definition 3.6), the input-output testing relation $\leq_{iot}$ (definition 4.1), and the input-output refusal relation $\leq_{ior}$ (definition 4.9): the traces and the completed traces of the composition of a process and an observer, expressed by $obs_t(u,p)$ and $obs_c(u,p)$, respectively. The varying parameters in defining these four relations were the distinction between inputs and outputs (and associated input-enabledness) and the ability to observe refusals by adding the θ-label to the class of observers.

Although all four relations were defined by requiring inclusion of both obs_c and of obs_t, some of the relations only need observations of one kind. This is indicated in table 1 by mentioning the necessary and sufficient observations. Adding the ability to observe refusals to observers, using the θ-action, makes observation of completed traces with obs_c superfluous: for $\leq_{rf}$ and $\leq_{ior}$ it suffices to consider observations of the kind obs_t. If no distinction between inputs and outputs is made, any observation of a trace in obs_t can always be simulated in obs_c with an observer which can perform only this particular trace and then terminates: for $\leq_{te}$ and $\leq_{rf}$ it

	$u \in \mathcal{LTS}(L)$	$u \in \mathcal{LTS}(L_\theta)$
no inputs and no outputs	$\leq_{te}$ obs_c	$\leq_{rf}$ obs_c^θ or obs_t^θ
inputs and outputs	$\leq_{iot}$ obs_c and obs_t	$\leq_{ior}$ obs_t^θ

Table 1: Observations obs_c and obs_t

suffices to consider observations of the kind obs_c. Only for $\leq_{iot}$ both kinds of observations are necessary, as shows the example in figure 5. Let $L_I = \emptyset$ and $L_U = \{x, y\}$; then, to define both intuitively incorrect implementations i_1 and i_2 as not $\leq_{iot}$-related, we need both obs_c and obs_t: $\forall u \in \mathcal{IOTS}(L_U, L_I) : obs_t(u, i_1) \subseteq obs_t(u, s)$, while $\forall u \in \mathcal{IOTS}(L_U, L_I) : obs_c(u, i_2) \subseteq obs_c(u, s)$.

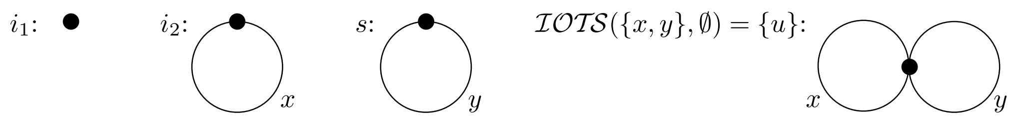

Figure 5: Observations for $\leq_{iot}$

The input-output implementation relations defined so far, viz. $\leq_{iot}$, **ioconf**, $\leq_{ior}$ and **ioco**, are easily related using their characterizations in terms of *out*-sets. The only difference between the relations is the set of (suspension) traces for which the *out*-sets are compared (cf. propositions 4.5 ($\leq_{iot}$) and 4.12 ($\leq_{ior}$), and definitions 4.7 (**ioconf**) and 4.13 (**ioco**)). So if we introduce the following class of relations $\mathbf{ioco}_\mathcal{F}$ with $\mathcal{F} \subseteq L_\delta^*$:

$$i \; \mathbf{ioco}_\mathcal{F} \; s \;\; =_{\mathrm{def}} \;\; \forall \sigma \in \mathcal{F} : \;\; out(\, i \; \mathbf{after} \; \sigma \,) \;\; \subseteq \;\; out(\, s \; \mathbf{after} \; \sigma \,) \tag{3}$$

then they can all be expressed as instances of $\mathbf{ioco}_\mathcal{F}$:

$$\begin{array}{lclclcl} \leq_{iot} & = & \mathbf{ioco}_{L^*} & \qquad & \mathbf{ioconf} & = & \mathbf{ioco}_{traces(s)} \\ \leq_{ior} & = & \mathbf{ioco}_{L_\delta^*} & & \mathbf{ioco} & = & \mathbf{ioco}_{Straces(s)} \end{array} \tag{4}$$

Using (3) and (4) the input-output implementation relations are easily related by relating their respective sets $\mathcal{F}$ (proposition 4.15). The inequalities follow from the candy machines in examples 4.8 and 4.14. The generalized implementation relation $\mathbf{ioco}_\mathcal{F}$ is the relation for which conformance testing and test derivation will be studied in section 6.

Proposition 4.15

$$\leq_{ior} \; \subset \; \left\{ \begin{array}{c} \leq_{iot} \\ \mathbf{ioco} \end{array} \right\} \; \subset \; \mathbf{ioconf}$$

□

4.4 Suspension automata

The characterizations of $\leq_{iot}$ in proposition 4.3 and of $\leq_{ior}$ in proposition 4.11 in terms of traces suggest to transform a labelled transition system into another one representing exactly the respective sets of traces, so that the relations can be characterized by trace preorder $\leq_{tr}$ (definition 3.1) on the results of this transformation. Such a transformed transition system can be obtained by taking the deterministic transition system representing exactly these sets of traces, if care is taken to correctly include possible quiescence. For $\leq_{ior}$ such a transition system is referred to as the *suspension automaton* Γ, and it is obtained from a transition system by adding loops $s \xrightarrow{\delta} s$ for all quiescent states and then determinizing the resulting automaton. It is easily seen that the implementation relation $\leq_{ior}$ reduces to trace preorder $\leq_{tr}$ on suspension automata. Moreover, *out*-sets can be directly transposed to suspension automata.

Definition 4.16
Let L be partitioned into L_I and L_U, and let $p = \langle S, L, T, s_0 \rangle \in \mathcal{LTS}(L)$ be a labelled transition system; then the *suspension automaton* of p, Γ_p, is the labelled transition system $\langle S_\delta, L_\delta, T_\delta, q_0 \rangle \in \mathcal{LTS}(L_\delta)$, where

- $S_\delta \;=_{\mathrm{def}}\; \mathcal{P}(S) \backslash \{\emptyset\}$ ($\mathcal{P}(S)$ is the powerset of S, i.e., the set of its subsets)
- $T_\delta \;=_{\mathrm{def}}\; \{\, q \xrightarrow{a} q' \mid a \in L_I \cup L_U,\ q, q' \in S_\delta,\ q' = \{ s' \in S \mid \exists s \in q : s \xRightarrow{a} s' \} \,\}$
  $\cup\ \{\, q \xrightarrow{\delta} q' \mid q, q' \in S_\delta,\ q' = \{ s \in q \mid \delta(s) \} \,\}$
- $q_0 \;=_{\mathrm{def}}\; \{\, s' \in S \mid s_0 \xRightarrow{\epsilon} s' \,\}$

□

Proposition 4.17
Let $p \in \mathcal{LTS}(L)$ with inputs in L_I and outputs in L_U, let $\sigma \in L_\delta^*$, and consider δ as an output action of Γ_p; i.e., Γ_p has inputs in L_I and outputs in $L_U \cup \{\delta\}$; then

1. Γ_p is deterministic.
2. $\mathit{traces}(\Gamma_p) \;=\; \mathit{Straces}(p)$
3. $\mathit{out}(\, \Gamma_p \,\mathbf{after}\, \sigma \,) \;=\; \mathit{out}(\, p \,\mathbf{after}\, \sigma \,)$
4. $\sigma \in \mathit{traces}(\Gamma_p)$ iff $\mathit{out}(\, \Gamma_p \,\mathbf{after}\, \sigma \,) \neq \emptyset$

□

Sketch of the proof
The first term of T_δ in definition 4.16 corresponds to the standard algorithm for determinization of automata. The second term adds δ-transitions for states in S_δ containing a quiescent state, thus precisely creating the suspension traces of p, while not affecting determinism. The third part follows from the fact that $\Gamma_p \,\mathbf{after}\, \sigma \;=\; p \,\mathbf{after}\, \sigma$, and the last part is clear from the construction of Γ_p: if there is no transition labelled with an output from $q \in S_\delta$, then there must be at least one quiescent state in q, so a δ-transition is added.

□

Corollary 4.18
$i \leq_{ior} s$ iff $\Gamma_i \leq_{tr} \Gamma_s$

□

So $\leq_{ior}$ is completely characterized by $\leq_{tr}$ on the corresponding suspension automata. Using (3) and (4) the other implementation relations can also be expressed in suspension automata restricting to suitable sets of traces, e.g., to $L^* \cdot \{\epsilon, \delta\}$ for $\leq_{iot}$. In [Tre96] a variant of the suspension automaton (called δ-trace automaton) was defined to characterize $\leq_{iot}$ directly. In that automaton, transitions of the form $q \xrightarrow{\delta} \mathbf{stop}$ were added for quiescent states, so that its traces are automatically restricted to $L^* \cdot \{\epsilon, \delta\}$.

Suspension automata are deterministic transition systems so that the transition relations $\xrightarrow{\sigma}$ and $\xRightarrow{\sigma}$ coincide, and each trace σ always goes to a unique state, denoted by $\Gamma_p \,\mathbf{after}\, \sigma$. The action δ, modelling quiescence, occurs as an explicit action in suspension automata. The action δ has all the properties of an output action. This leads us to the conclusion that input-output transition systems can be considered modulo trace equivalence if quiescence is added as an additional, observable output action. Because of these properties, the suspension automaton of a specification will be the

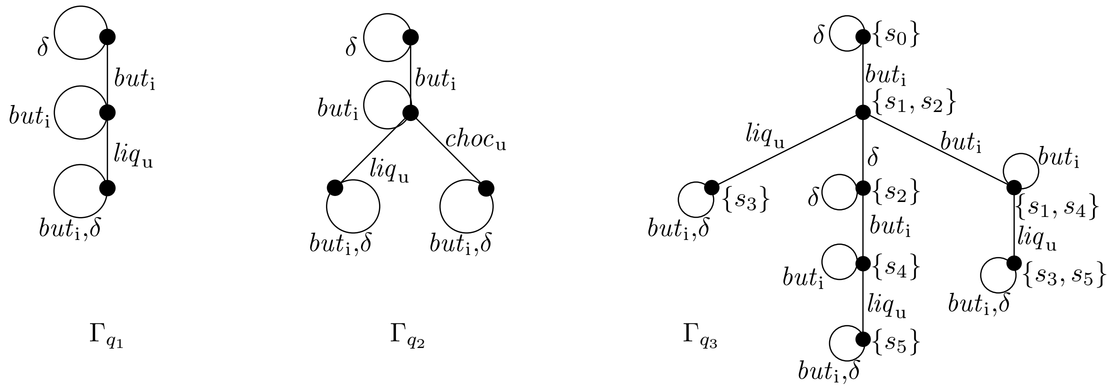

Figure 6: Suspension automata for figure 1

basis for the derivation of tests in section 6.

Example 4.19
Figures 6 and 7 show the suspension automata for the processes of figures 1 and 4, respectively. For Γ_{q_3} the states, subsets of states of q_3, have been added. Note that the nondeterminism of q_3 is removed and that state $\{s_1, s_2\}$ has a δ-transition, since there is a state in $\{s_1, s_2\}$, i.e. s_2, that refuses all outputs. Using corollary 4.18 we can now easily check that $r_2 \leq_{ior} r_1$, but $r_1 \not\leq_{ior} r_2$ (example 4.14).

□

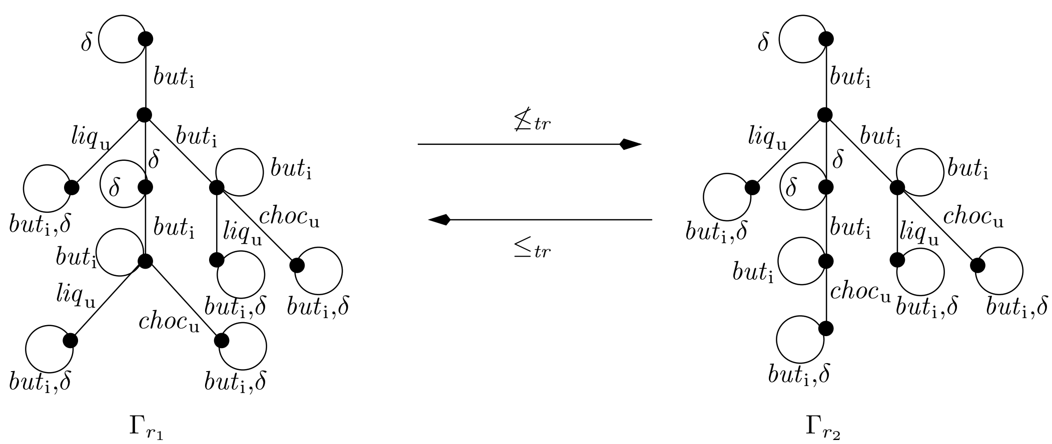

Figure 7: Suspension automata for figure 4

5 Testing Input-Output Transition Systems

Now that we have formal specifications (expressed as labelled transition systems), implementations (modelled by input-output transition systems), and a formal definition of conformance (expressed by one of the implementation relations $\mathbf{ioco}_{\mathcal{F}}$), we can start the discussion of conformance testing. This means that the statement of (1) is reversed: instead of defining an implementation relation by choosing a set of observers, we look for a minimal (or at least reduced) set of observers or test cases which fully characterizes all $\mathbf{ioco}_{\mathcal{F}}$-correct implementations of a given specification. The first point of discussion is what these test cases look like, how they are executed, and when they are successful.

A test case is a specification of the behaviour of a tester in an experiment to be carried out with an implementation under test. Such behaviour, like other behaviours, can be described by a labelled transition system. In particular, since we consider the relations $\mathbf{ioco}_{\mathcal{F}}$, tests will be processes in $\mathcal{LTS}(L_I \cup L_U \cup \{\theta\})$. But to guarantee that the experiment lasts for a finite time, a test case should have finite behaviour. Moreover, a tester executing a test case would like to have control over the testing process as much as possible, so a test case should be specified in such a way that unnecessary nondeterminism is avoided. First of all, this implies that the test case itself must be deterministic. But also we will not allow test cases with a choice between an input action and an output action, nor a choice between multiple input actions (input and output from the perspective of the implementation). Both introduce unnecessary nondeterminism in the test run: if a test case can offer multiple input actions, or a choice between input and output, then the continuation of the test run is unnecessarily nondeterministic, since any input-output implementation can always accept any input action. This implies that a state of a test case either is a terminal state, or offers one particular input to the implementation, or accepts all possible outputs from the implementation, including the δ-action which is accepted by a θ-action in the test case. Finally, to be able to decide about the success of a test, the terminal states of a test case are labelled with the verdict **pass** or **fail**. Altogether, we come to the following definition of a test case.

Definition 5.1

1. A *test case* t is a labelled transition system $\langle S, L_I \cup L_U \cup \{\theta\}, T, s_0 \rangle$ such that
   - t is deterministic and has finite behaviour;
   - S contains the terminal states **pass** and **fail**, with $init(\mathbf{pass}) = init(\mathbf{fail}) = \emptyset$;
   - for any state $t' \in S$ of the test case, $t' \neq \mathbf{pass}, \mathbf{fail}$, either $init(t') = \{a\}$ for some $a \in L_I$, or $init(t') = L_U \cup \{\theta\}$.

   The class of test cases over L_U and L_I is denoted as $\mathcal{TEST}(L_U, L_I)$.
2. A *test suite* T is a set of test cases: $T \subseteq \mathcal{TEST}(L_U, L_I)$.

□

Note that L_I and L_U refer to the inputs and outputs from the point of view of the implementation under test, so L_I denotes the outputs, and L_U denotes the inputs of the test case.

A test run of an implementation with a test case is modelled by the synchronous parallel execution $\rceil|$ of the test case with the implementation under test, which continues until no more interactions are possible, i.e., until a deadlock occurs (definition 3.2). Since for each state $t' \neq \mathbf{pass}, \mathbf{fail}$ of a test case either $init(t') = \{a\}$ for some $a \in L_I$, in which case no deadlock can occur because of input-enabledness of implementations, or $init(t') = L_U \cup \{\theta\}$, in which case no deadlock can occur because of the θ-transition, it follows that deadlock can only occur in the terminal states **pass** and **fail**. An implementation passes a test run if and only if the test run deadlocks in **pass**. Since an implementation can behave nondeterministically, different test runs of the same test case with the same implementation may lead to different terminal states and hence to different verdicts. An implementation passes a test case if and only if all possible test runs lead to the verdict **pass**.

This means that each test case must be executed several times in order to give a final verdict, theoretically even infinitely many times.

Definition 5.2

1. A *test run* of a test case $t \in \mathcal{TEST}(L_U, L_I)$ with an implementation $i \in \mathcal{IOTS}(L_I, L_U)$ is a trace of the synchronous parallel composition $t\rceil\!|\,i$ leading to a terminal state of t:

$$\sigma \text{ is a test run of } t \text{ and } i \;\; =_{\text{def}} \;\; \exists i' : \; t\rceil\!|\,i \overset{\sigma}{\Longrightarrow} \textbf{pass}\rceil\!|\,i' \;\text{ or }\; t\rceil\!|\,i \overset{\sigma}{\Longrightarrow} \textbf{fail}\rceil\!|\,i'$$

2. An implementation i *passes* a test case t if all their test runs lead to the **pass**-state of t:

$$i \;\textbf{passes}\; t \;\; =_{\text{def}} \;\; \forall \sigma \in L_\theta^*,\; \forall i' : \; t\rceil\!|\,i \overset{\sigma}{\not\Longrightarrow} \textbf{fail}\rceil\!|\,i'$$

3. An implementation i *passes* a test suite T if it passes all test cases in T:

$$i \;\textbf{passes}\; T \;\; =_{\text{def}} \;\; \forall t \in T : \; i \;\textbf{passes}\; t$$

If i does not pass the test suite, it fails: i **fails** $T =_{\text{def}} \exists t \in T : i \not\textbf{passes}\; t$.

□

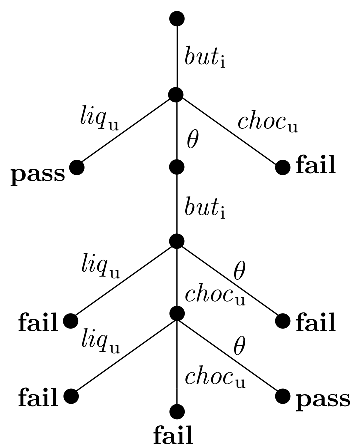

Figure 8: A test case

Example 5.3
For r_1 (figure 4) there are three test runs with t in figure 8:

$$t\rceil\!|\,r_1 \xRightarrow{but_\text{i}\cdot liq_\text{u}} \textbf{pass}\rceil\!|\,r_1'$$
$$t\rceil\!|\,r_1 \xRightarrow{but_\text{i}\cdot\theta\cdot but_\text{i}\cdot liq_\text{u}} \textbf{fail}\rceil\!|\,r_1''$$
$$t\rceil\!|\,r_1 \xRightarrow{but_\text{i}\cdot\theta\cdot but_\text{i}\cdot choc_\text{u}\cdot\theta} \textbf{pass}\rceil\!|\,r_1'''$$

where r_1', r_1'', and r_1''' are the leaves of r_1 from left to right. Since the terminal state of t for the second run is **fail**, we have r_1 **fails** t. Similarly, it can be checked that r_2 **passes** t.

□

6 Test Generation for Input-Output Transition Systems

Now all ingredients are there to present an algorithm to generate test suites from labelled transition system specifications for implementation relations of the form $\mathbf{ioco}_{\mathcal{F}}$. A generated test suite must test implementations for conformance with respect to s and $\mathbf{ioco}_{\mathcal{F}}$. Ideally, an implementation should pass the test suite if and only if it conforms. In this case the test suite is called *complete* [ISO96]. Unfortunately, in almost all practical cases such a test suite would be infinitely large; hence for practical testing we have to restrict to test suites that can only detect non-conformance, but that cannot assure conformance. Such test suites are called *sound*. Test suites that can only assure conformance but that may also reject conforming implementations are called exhaustive.

Definition 6.1
Let s be a specification and T a test suite; then for an implementation relation $\mathbf{ioco}_{\mathcal{F}}$:

$$\begin{array}{llllll} T \text{ is complete} & =_{\text{def}} & \forall i: & i \ \mathbf{ioco}_{\mathcal{F}} \ s & \text{iff} & i \ \mathbf{passes} \ T \\ T \text{ is sound} & =_{\text{def}} & \forall i: & i \ \mathbf{ioco}_{\mathcal{F}} \ s & \text{implies} & i \ \mathbf{passes} \ T \\ T \text{ is exhaustive} & =_{\text{def}} & \forall i: & i \ \mathbf{ioco}_{\mathcal{F}} \ s & \text{if} & i \ \mathbf{passes} \ T \end{array}$$

□

We aim at producing sound test suites from a specification s, and for that purpose we use the suspension automaton Γ_s. To get some idea what such test cases will look like, we consider the definition of **ioco** in terms of suspension automata (definition 4.13 with proposition 4.17.3). We see that to test for **ioco** we have to check for each $\sigma \in \mathit{traces}(\Gamma_s)$ whether $\mathit{out}(\, \Gamma_i \ \mathbf{after}\, \sigma\,) \subseteq \mathit{out}(\, \Gamma_s \ \mathbf{after}\, \sigma\,)$. Basically, this can be done by having a test case t that executes σ:

$$t\rceil|\, i \stackrel{\sigma}{\Longrightarrow} t'\rceil|\, i'$$

and then checks $\mathit{out}(\, \Gamma_i \ \mathbf{after}\, \sigma\,)$ by having transitions to **pass**-states for all allowed outputs (those in $\mathit{out}(\, \Gamma_s \ \mathbf{after}\, \sigma\,)$) and transitions to **fail**-states for all erroneous outputs (those not in $\mathit{out}(\, \Gamma_s \ \mathbf{after}\, \sigma\,)$). Special care should be taken for the special output δ: δ actually models the absence of any output, so no transition will be made by the implementation if i' 'outputs' δ. This matches a θ-transition in the test case, which exactly occurs if nothing else can happen. This θ-transition will go the **pass**-state if quiescence is allowed ($\delta \in \mathit{out}(\, \Gamma_s \ \mathbf{after}\, \sigma\,)$) and to the **fail**-state if the specification does not allow quiescence at that point. All this is reflected in the following recursive algorithm for test generation for relations $\mathbf{ioco}_{\mathcal{F}}$ with $\mathcal{F} \subseteq \mathit{Straces}(s)$. The algorithm is nondeterministic in the sense that in each recursive step it can be continued in many different ways: termination of the test case in choice 1, any input action satisfying the requirement of choice 2, or checking the allowed outputs in choice 3. Each continuation will result in another sound test case (theorem 6.3.1), and all possible test cases together form an exhaustive (and thus complete) test suite (theorem 6.3.2), so there are no errors in an implementation that are principally undetectable with test suites generated with the algorithm. However, if the behaviour of the specification is infinite, the algorithm allows construction of infinitely many different test cases, which can be arbitrarily long, but which all have finite behaviour.

In the algorithm we use the notation $\overline{\sigma}$ for a trace in which all δ-actions have been replaced by θ-actions (and others left unchanged), or vice versa, depending on the domain of σ. The interchange of δ- and θ-actions is natural since they are in a certain sense complementary: δ models the absence of outputs (the refusal L_U) which can be observed with θ and which is the only refusal which can be observed when dealing with input-output transition systems (cf. complementary actions a and $\overline{a}$ in CCS [Mil89]).

Algorithm 6.2
Let Γ be the suspension automaton of a specification, and let $\mathcal{F} \subseteq traces(\Gamma)$; then a test case $t \in \mathcal{TEST}(L_U, L_I)$ is obtained by a finite number of recursive applications of one of the following three nondeterministic choices:

1. (∗ terminate the test case ∗)
   $t \; := \; \textbf{pass}$
2. (∗ give a next input to the implementation ∗)
   $t \; := \; a \; ; \; t'$
   where $a \in L_I$, such that $\mathcal{F}' = \{\sigma \in L_\delta^* \mid a{\cdot}\sigma \in \mathcal{F}\} \neq \emptyset$, and t' is obtained by recursively applying the algorithm for $\mathcal{F}'$ and Γ', with $\Gamma \xrightarrow{a} \Gamma'$.
3. (∗ check the next output of the implementation ∗)
   $$\begin{array}{lll} t \; := & & \Sigma \, \{ \, x \; ; \; \textbf{fail} \mid x \in L_U \cup \{\theta\}, \; \overline{x} \notin out(\Gamma), \; \epsilon \in \mathcal{F} \} \\ & \Box & \Sigma \, \{ \, x \; ; \; \textbf{pass} \mid x \in L_U \cup \{\theta\}, \; \overline{x} \notin out(\Gamma), \; \epsilon \notin \mathcal{F} \} \\ & \Box & \Sigma \, \{ \, x \; ; \; t_x \mid x \in L_U \cup \{\theta\}, \; \overline{x} \in out(\Gamma) \; \} \end{array}$$
   where t_x is obtained by recursively applying the algorithm for $\{\sigma \in L_\delta^* \mid \overline{x}{\cdot}\sigma \in \mathcal{F}\}$ and Γ', with $\Gamma \xrightarrow{\overline{x}} \Gamma'$.

□

Theorem 6.3
Let $s \in \mathcal{LTS}(L_I \cup L_U)$ and $\mathcal{F} \subseteq Straces(s)$; then

1. a test case obtained with algorithm 6.2 from Γ_s and $\mathcal{F}$ is sound for s with respect to $\textbf{ioco}_\mathcal{F}$;
2. the set of all possible test cases that can be obtained with algorithm 6.2 is exhaustive.

□

Sketch of the proof

1. Soundness can be proved using the following sufficient condition for soundness of a test case $t \in \mathcal{TEST}(L_U, L_I)$ for a specification s with respect to $\textbf{ioco}_\mathcal{F}$:

   $$\begin{array}{ll} \forall \sigma \in L_\theta^* : & t \xrightarrow{\sigma} \textbf{fail} \;\text{ implies} \\ & \exists \sigma' \in \mathcal{F}, \; x \in L_U \cup \{\delta\} : \; \sigma = \overline{\sigma' {\cdot} x} \;\text{ and }\; x \notin out(\,\Gamma_s \,\textbf{after}\, \sigma'\,) \end{array} \qquad (5)$$

   This property is proved by contradiction: let t be not sound, then $\exists i$: $i \;\textbf{ioco}_\mathcal{F}\; s$, and $t \rceil\!\lceil i \xRightarrow{\sigma} \textbf{fail} \rceil\!\lceil i'$. It follows that $t \xrightarrow{\sigma} \textbf{fail}$ and $i \xRightarrow{\overline{\sigma}} i'$, so from the premiss: $\exists \sigma' \in \mathcal{F}$, $x \in L_U \cup \{\delta\}$: $\sigma = \overline{\sigma' {\cdot} x}$ and $x \notin out(\,\Gamma_s \,\textbf{after}\, \sigma'\,)$. But since $i \xRightarrow{\sigma' \cdot x} i'$ and $i \;\textbf{ioco}_\mathcal{F}\; s$, we have $x \in out(\,\Gamma_s \,\textbf{after}\, \sigma'\,)$, so a contradiction.

   By straightforward induction on the structure of t, it is then proved that each t generated with algorithm 6.2 from Γ_s and $\mathcal{F}$ satisfies property (5).

2. For exhaustiveness we have to show that the set of all test cases $\mathcal{T}$ generated with algorithm 6.2 satisfies $\forall i : i \;\cancel{\textbf{ioco}}_\mathcal{F}\; s$ implies $\exists t \in \mathcal{T} : i \;\textbf{fails}\; t$. So let $\sigma \in \mathcal{F}$ such that $out(\,i \,\textbf{after}\, \sigma\,) \not\subseteq out(\,s \,\textbf{after}\, \sigma\,)$, so $\exists z \in out(\,i \,\textbf{after}\, \sigma\,)$ with $z \notin out(\,s \,\textbf{after}\, \sigma\,)$. A test case $t_{[\sigma]}$ can be constructed with algorithm 6.2 from Γ_s and $\mathcal{F}$ as follows:

   - $t_{[\epsilon]}$ is obtained with the third choice in algorithm 6.2, followed by the first choice for each t_x;
   - $t_{[b \cdot \sigma]}$ ($b \in L_I$) is obtained with the second choice in algorithm 6.2, choosing $a = b$, and followed by recursive application to obtain $t' = t_{[\sigma]}$;
   - $t_{[\overline{y} \cdot \sigma]}$ ($y \in L_U \cup \{\theta\}$) is obtained with the third choice in algorithm 6.2, followed by the first choice for each t_x with $x \neq y$, and recursive application to obtain $t_y = t_{[\sigma]}$.

   Now it can be shown that $t_{[\sigma]} \rceil\!\lceil i \xRightarrow{\overline{\sigma}} t_{[\epsilon]} \rceil\!\lceil i' \xrightarrow{\overline{z}} \textbf{fail} \rceil\!\lceil i''$, so i **fails** $t_{[\sigma]}$.

□

Example 6.4
Consider the candy machines r_1 and r_2 in figure 4. We use algorithm 6.2 to derive a test case from r_2 with respect to $\mathbf{ioco} = \mathbf{ioco}_{Straces(s)} = \mathbf{ioco}_{traces(\Gamma_s)}$. In the derivation we use the suspension automaton of figure 7. The successive choices for the recursive steps of the algorithm are:

1. First choice 2 (giving an input to the implementation) is taken: $t := but_{\mathrm{i}}; t_1$
2. To obtain t_1, the next output of the implementation is checked (choice 3): $t_1 \ := \ liq_{\mathrm{u}}; t_{2_1} \ \Box \ choc_{\mathrm{u}}; \mathbf{fail} \ \Box \ \theta; t_{2_2}$
3. If the output was liq_{u}, then we stop with the test case (choice 1): $t_{2_1} := \mathbf{pass}$
4. If no output was produced (output θ; we know that we are in the right branch of r_2), then we give a next input to the implementation (choice 2): $t_{2_2} \ := \ but_{\mathrm{i}}; t_3$
5. To obtain t_3 we again check the outputs (choice 3): $t_3 \ := \ choc_{\mathrm{u}}; t_4 \ \Box \ liq_{\mathrm{u}}; \mathbf{fail} \ \Box \ \theta; \mathbf{fail}$
6. After the output $choc_{\mathrm{u}}$ we check again the outputs (choice 3) to be sure that nothing more is produced: $t_4 \ := \ choc_{\mathrm{u}}; \mathbf{fail} \ \Box \ liq_{\mathrm{u}}; \mathbf{fail} \ \Box \ \theta; t_5$
7. For t_5 we stop with the test case (choice 1): $t_5 := \mathbf{pass}$

After putting together all pieces, we obtain t of figure 8 as a sound test case for r_2, which is consistent with the results in examples 4.14 and 5.3: $r_1 \ \mathbf{io\not{c}o} \ r_2$, $r_2 \ \mathbf{ioco} \ r_2$, and indeed r_1 **fails** t, and r_2 **passes** t. So test case t will detect that r_1 is not **ioco**-correct with respect to r_2.

$\Box$

Algorithm 6.2 is restricted to implementation relations $\mathbf{ioco}_{\mathcal{F}}$ and specifications s such that $\mathcal{F} \subseteq traces(\Gamma_s)$. This looks more restrictive than it is. First, proposition 6.5 below shows that in the case of $\mathcal{F}$ being prefix-closed (for each $\sigma \in \mathcal{F}$ also all its prefixes are in $\mathcal{F}$) it suffices to consider only the suspension traces of s and the suspension traces of s concatenated with one input action. If an implementation contains an error (i.e., $out(\, i \,\mathbf{after}\, \sigma \,) \not\subseteq out(\, s \,\mathbf{after}\, \sigma \,)$) for a trace σ which is not a suspension trace or a suspension trace concatenated with one input action, then there is always a prefix of σ which is such a trace and which leads to an error, too.

Secondly, a form of inconsistency between the specification s and the set of traces $\mathcal{F}$ occurs if $Straces(s) \neq Straces(s) \cdot L_I$. Suppose there is a trace $\sigma \cdot x \cdot a \in \mathcal{F}$ such that $\sigma \cdot x$ is a suspension trace of s but $\sigma \cdot x \cdot a$ is not. Then $\Gamma_s \xrightarrow{\sigma \cdot x} \Gamma'$ where Γ' is not input enabled: $\exists a \in L_I : \Gamma' \overset{a}{\not\longrightarrow}$. The consequence is that we have $x \in out(\, s \,\mathbf{after}\, \sigma \,)$, so a conforming implementation may also perform x after σ: $x \in out(\, i \,\mathbf{after}\, \sigma \,)$. However, then $i \overset{\sigma \cdot x}{\Longrightarrow}$ and since i is an input-output transition system, also $i \overset{\sigma \cdot x \cdot a}{\Longrightarrow}$. So we have $out(\, i \,\mathbf{after}\, \sigma \cdot x \cdot a \,) \neq \emptyset$, while $out(\, s \,\mathbf{after}\, \sigma \cdot x \cdot a \,) = \emptyset$, so i cannot be a conforming implementation of s (cf. the discussion on input-enabledness and non-implementability of specifications in section 4.1). Combinations of s and $\mathcal{F}$ leading to this problem do not make sense and, if really needed, can always be replaced by modified s and $\mathcal{F}$ not having this problem and defining the same class of implementations. Consequently, since usually $\mathcal{F}$ is prefix-closed, it can be replaced by $\mathcal{G} \subseteq Straces(s)$, so, algorithm 6.2 has a useful and broad range of applicability.

Proposition 6.5
Let $\mathcal{F}$ be prefix-closed and let $\mathcal{G} = \mathcal{F} \cap (Straces(s) \cup Straces(s) \cdot L_I)$; then $\mathbf{ioco}_{\mathcal{F}} = \mathbf{ioco}_{\mathcal{G}}$

$\Box$

7 Concluding Remarks

This paper has presented implementation relations, conformance testing and test generation for implementations that communicate via inputs and outputs. The implementation relations were defined by applying the theory of testing equivalences and refusal testing to systems in which inputs and outputs can be distinguished and in which inputs are always enabled. The defined relations $\leq_{iot}$, $\leq_{ior}$, **ioconf** and **ioco** are particular instances of a class of implementation relations $\mathbf{ioco}_{\mathcal{F}}$, which are most easily characterized if the refusal of outputs, i.e., quiescence, is considered as an explicitly observable event represented by a special output action δ. Traces over input actions, outputs actions and δ are called suspension traces, and the parameter $\mathcal{F}$ in $\mathbf{ioco}_{\mathcal{F}}$ is a set of them. Processes modulo these input-output implementation relations are fully characterized by (a subset of) their suspension traces. The action δ is no different from a normal output action. This means that the addition of δ to represent quiescence makes it possible to reason about systems using only linear properties, i.e., traces.

The characterization in terms of suspension traces formed the basis for a test generation algorithm which was proved to derive test cases from a specification, which can detect, by means of conformance testing, all and only implementations which are incorrect for that specification with respect to any of the implementation relations $\mathbf{ioconf}_{\mathcal{F}}$. The resulting theory and algorithm are somewhat simpler than the corresponding theory and algorithms for testing with symmetric interactions (e.g., compare proposition 4.3 with 3.3, and compare algorithm 6.2 with the **conf**-based test generation algorithm in [Tre92]). The theory and the algorithm can form the basis for the development of test generation tools. They can be applied to those domains where implementations can be assumed to communicate via inputs and outputs, which is the case for many realistic systems, and where specifications can be expressed in labelled transition systems, which also holds for many specification formalisms.

It was indicated that input-output transition systems only marginally differ from Input/Output Automata [LT89] in that the former has a weaker requirement on input-enabling. This allows for some systems that are $\mathcal{IOTS}$ but not IOA (e.g., communication with systems via explicitly modelled bounded buffers: when the buffer is full, no input actions are possible anymore without first performing an internal event. Such a system is $\mathcal{IOTS}$.).

Another model which is very much related to input-output transition systems, is that of Input-Output State Machines (IOSM) [Pha94]. Our suspension automaton was inspired by the way the absence of output is treated in [Pha94]. Like IOA, IOSMs have strong input-enabling (called completeness). Absence of outputs (*'blocage de sortie'*) is also considered observable, and an implementation relation R_1 is defined which strongly resembles **ioco**.

The interesting point about the relation R_1 is that it was defined without reference to an underlying theory of testing equivalence or refusal testing, but that it was defined as the result of formalizing existing protocol testing practice with an existing testing tool (TVEDA [CGPT96]) based on formal specifications in Estelle and SDL. This may be an indication that relations like **ioco** have not only a nice theoretical basis but also have practical applicability. A first trial to apply the theory of **ioconf** to conformance testing of a very simple protocol looks promising [TFPHT96].

The implementation relations and algorithm in this paper generalize those for queue systems [TV92]. Queue systems are transition systems in a queue context, i.e., to which two unbounded queues are attached to model asynchronous communication, one queue for inputs and one for outputs. An unbounded queue clearly has the property that input can never be refused, while the output queue determines that from the system's point of view output actions can never be refused by the environment.

An open issue is the atomicity of actions. Although we allow specifications to be labelled transition systems, the actions are classified as inputs and outputs, and they have a one-to-one correspondence to those of the implementation. An interesting area for further investigation occurs if implementation relations are combined with action refinement, so that one abstract symmetric interaction of the specification is implemented using multiple inputs and outputs, e.g., implementing an abstract interaction by means of a handshake protocol. Tests could be derived from the specification using symmetric algorithms (section 3) and then refined, or the specification could be refined after which the input-output-based algorithm is used. The precise relation between testing, inputs and outputs, and action refinement needs further investigation.

Adding the absence of outputs as a special observable event makes it easier to compare transition systems with other models in which the absence of outputs is treated in the same way, such as in the realm of Mealy Machines (Finite State Machines FSM). FSMs are often used in the area of communication protocol conformance testing [BU91, YL95], where the absence of outputs is usually denoted by a special *null-output*. The precise relation between the testing theories based on labelled transition systems and those based on FSMs is left for further study.

More attention is also needed to the topic of efficiently and effectively obtaining suspension automata and test cases. In particular, a compositional method for deriving them from process-algebraic specifications would be helpful. Also, equational and congruence properties and axiomatization are left for further study.

Among the more practically oriented problems are the well-known test selection problem (test-suite size reduction [ISO96]). Algorithm 6.2 can generate infinitely many sound test cases, but which ones shall be really executed? Solutions can be sought by defining coverage measures, fault models, stronger test hypotheses, etc. [Ber91, ISO96, Pha94, Tre92]. Another aspect is the incorporation of data in the test generation procedure. The state explosion caused by the data in specifications needs to be handled in a symbolic way; otherwise automation of the test generation algorithm in test tools will probably not be feasible. A last practical problem is the implementation of the observation of quiescence. In practical test execution tools, timers will have to be used for which the time-out values need to be chosen carefully in order not to observe quiescence where there is none.

References

[Abr87] S. Abramsky. Observational equivalence as a testing equivalence. *Theoretical Computer Science*, 53(3):225–241, 1987.

[BAL+90] E. Brinksma, R. Alderden, R. Langerak, J. van de Lagemaat, and J. Tretmans. A formal approach to conformance testing. In J. de Meer, L. Mackert, and W. Effelsberg, editors, *Second Int. Workshop on Protocol Test Systems*, pages 349–363. North-Holland, 1990.

[Ber91] G. Bernot. Testing against formal specifications: A theoretical view. In S. Abramsky and T. S. E. Maibaum, editors, *TAPSOFT'91, Volume 2*, pages 99–119. Lecture Notes in Computer Science 494, Springer-Verlag, 1991.

[BK85] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[Bri88] E. Brinksma. A theory for the derivation of tests. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing, and Verification VIII*, pages 63–74. North-Holland, 1988.

[BSS87] E. Brinksma, G. Scollo, and C. Steenbergen. LOTOS specifications, their implementations and their tests. In G. von Bochmann and B. Sarikaya, editors, *Protocol Specification, Testing, and Verification VI*, pages 349–360. North-Holland, 1987.

[BU91] B. S. Bosik and M. Ü. Uyar. Finite state machine based formal methods in protocol conformance testing: From theory to implementation. *Computer Networks and ISDN Systems*, 22(1):7–33, 1991.

[CCI92] CCITT. *Specification and Description Language (SDL).* Recommendation Z.100. ITU-T General Secretariat, Geneve, Switzerland, 1992.

[CGPT96] M. Clatin, R. Groz, M. Phalippou, and R. Thummel. Two approaches linking test generation with verification techniques. In A. Cavalli and S. Budkowski, editors, *Eight Int. Workshop on Protocol Test Systems*. Chapman & Hall, 1996.

[DN87] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24:211–237, 1987.

[DNH84] R. De Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[DNS95] R. De Nicola and R. Segala. A process algebraic view of Input/Output Automata. *Theoretical Computer Science*, 138:391–423, 1995.

[Gla90] R.J. van Glabbeek. The linear time – branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR'90*, Lecture Notes in Computer Science 458, pages 278–297. Springer-Verlag, 1990.

[Gla93] R.J. van Glabbeek. The linear time – branching time spectrum II (The semantics of sequential systems with silent moves). In E. Best, editor, *CONCUR'93*, Lecture Notes in Computer Science 715, pages 66–81. Springer-Verlag, 1993.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[ISO89a] ISO. *Information Processing Systems, Open Systems Interconnection, Estelle - A Formal Description Technique Based on an Extended State Transition Model.* International Standard IS-9074. ISO, Geneve, 1989.

[ISO89b] ISO. *Information Processing Systems, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. International Standard IS-8807. ISO, Geneve, 1989.

[ISO96] ISO/IEC JTC1/SC21 WG7, ITU-T SG 10/Q.8. *Information Retrieval, Transfer and Management for OSI; Framework: Formal Methods in Conformance Testing*. Committee Draft CD 13245-1, ITU-T proposed recommendation Z.500. ISO – ITU-T, Geneve, 1996.

[Lan90] R. Langerak. A testing theory for LOTOS using deadlock detection. In E. Brinksma, G. Scollo, and C. A. Vissers, editors, *Protocol Specification, Testing, and Verification IX*, pages 87–98. North-Holland, 1990.

[Led92] G. Leduc. A framework based on implementation relations for implementing LOTOS specifications. *Computer Networks and ISDN Systems*, 25(1):23–41, 1992.

[LT89] N.A. Lynch and M.R. Tuttle. An introduction to Input/Output Automata. *CWI Quarterly*, 2(3):219–246, 1989.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[PF90] D. H. Pitt and D. Freestone. The derivation of conformance tests from LOTOS specifications. *IEEE Transactions on Software Engineering*, 16(12):1337–1343, 1990.

[Pha94] M. Phalippou. *Relations d'Implantation et Hypothèses de Test sur des Automates à Entrées et Sorties*. PhD thesis, L'Université de Bordeaux I, France, 1994.

[Phi87] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50(2):241–284, 1987.

[Seg93] R. Segala. Quiescence, fairness, testing, and the notion of implementation. In E. Best, editor, *CONCUR'93*, pages 324–338. Lecture Notes in Computer Science 715, Springer-Verlag, 1993.

[TFPHT96] R. Terpstra, L. Ferreira Pires, L. Heerink, and J. Tretmans. Testing theory in practice: A simple experiment. In T. Kapus and Z. Brezočnik, editors, *COST 247 Int. Workshop on Applied Formal Methods in System Design*, pages 168–183, Maribor, Slovenia, 1996. University of Maribor.

[Tre92] J. Tretmans. *A Formal Approach to Conformance Testing*. PhD thesis, University of Twente, Enschede, The Netherlands, 1992.

[Tre96] J. Tretmans. Test generation with inputs, outputs, and quiescence. In T. Margaria and B. Steffen, editors, *Second Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 127–146. Lecture Notes in Computer Science 1055, Springer-Verlag, 1996.

[TV92] J. Tretmans and L. Verhaard. A queue model relating synchronous and asynchronous communication. In R.J. Linn and M.Ü. Uyar, editors, *Protocol Specification, Testing, and Verification XII*, number C-8 in IFIP Transactions, pages 131–145. North-Holland, 1992.

[Vaa91] F. Vaandrager. On the relationship between process algebra and Input/Output Automata. In *Logic in Computer Science*, pages 387–398. Sixth Annual IEEE Symposium, IEEE Computer Society Press, 1991.

[Wez90] C. D. Wezeman. The CO-OP method for compositional derivation of conformance testers. In E. Brinksma, G. Scollo, and C. A. Vissers, editors, *Protocol Specification, Testing, and Verification IX*, pages 145–158. North-Holland, 1990.

[YL95] M. Yannakakis and D. Lee. Testing finite state machines: Fault detection. *Journal of Computer and System Sciences*, 50(2):209–227, 1995.

Acknowledgements

Stimulating discussions with and comments from the Formal-Methods people within the Tele-Informatics and Open Systems Group greatly helped in writing this paper. Also, the comments from participants in the European Research Action COST 247 'Verification and Validation Methods for Formal Descriptions', from anonymous reviewers, and from TACAS'96 participants are acknowledged.

A Proofs

A.1 Proofs of Section 3 (Implementation relations with symmetric interactions)

We start with proving the claim of equation (2). Then some lemmata are presented for the proofs of propositions 3.3 and 3.5: lemma A.1 investigates the behaviour of the parallel operator $\|$, lemma A.2.1 relates observations to failure pairs, lemma A.2.2 defines a special observer which can observe a particular failure pair, lemma A.3 shows that observations of the kind obs_t are not necessary, and finally, lemma A.4 relates observations of kind obs_c to failure pairs. Propositions 3.3 and 3.5 are then direct consequences of these lemmata.

Claim (2)

p **after** σ **refuses** A iff $\exists p' : p \overset{\sigma}{\Longrightarrow} p'$ and $\forall \mu \in A \cup \{\tau\} : p' \overset{\mu}{\not\longrightarrow}$ □

Proof (claim (2))

only if:

$$
\begin{array}{ll}
 & p \textbf{ after } \sigma \textbf{ refuses } A \\
\text{implies} & (*\ \text{definition 3.2.1}\ *) \\
 & \exists p' : p \overset{\sigma}{\Longrightarrow} p' \text{ and } \forall a \in A : p' \overset{a}{\not\Longrightarrow} \\
\text{implies} & (*\ \text{strong convergence}\ *) \\
 & \exists p' : p \overset{\sigma}{\Longrightarrow} p' \text{ and } \forall a \in A : p' \overset{a}{\not\Longrightarrow} \text{ and } \exists p'' : p' \overset{\epsilon}{\Longrightarrow} p'' \overset{\tau}{\not\longrightarrow} \\
\text{implies} & (*\ \text{definition 2.2}\ *) \\
 & \exists p', p'' : p \overset{\sigma}{\Longrightarrow} p' \overset{\epsilon}{\Longrightarrow} p'' \text{ and } \forall a \in A : p'' \overset{a}{\not\longrightarrow} \text{ and } p'' \overset{\tau}{\not\longrightarrow} \\
\text{implies} & (*\ \text{definition 2.2}\ *) \\
 & \exists p'' : p \overset{\sigma}{\Longrightarrow} p'' \text{ and } \forall \mu \in A \cup \{\tau\} : p'' \overset{\mu}{\not\longrightarrow}
\end{array}
$$

if:

$$
\begin{array}{ll}
 & \exists p' : p \overset{\sigma}{\Longrightarrow} p' \text{ and } \forall \mu \in A \cup \{\tau\} : p' \overset{\mu}{\not\longrightarrow} \\
\text{implies} & (*\ \text{definition 2.2}\ *) \\
 & \exists p' : p \overset{\sigma}{\Longrightarrow} p' \text{ and } \forall a \in A : p' \overset{a}{\not\Longrightarrow} \\
\text{implies} & (*\ \text{definition 3.2.1}\ *) \\
 & p \textbf{ after } \sigma \textbf{ refuses } A
\end{array}
$$

□

Lemma A.1

Let $p, q, r \in \mathcal{LTS}(L)$, $\sigma \in L^*$, then

1. $p \| q \overset{\sigma}{\Longrightarrow} r$ implies $\exists p', q' : p \overset{\sigma}{\Longrightarrow} p'$ and $q \overset{\sigma}{\Longrightarrow} q'$ and $r = p' \| q'$
2. $p \overset{\sigma}{\Longrightarrow} p'$ and $q \overset{\sigma}{\Longrightarrow} q'$ imply $p \| q \overset{\sigma}{\Longrightarrow} p' \| q'$

□

Proof (lemma A.1)

1. By induction on the structure of σ:

   $\sigma = \epsilon$: The lemma reduces to

$$p \| q \overset{\epsilon}{\Longrightarrow} r \quad \text{implies} \quad \exists p', q' : p \overset{\epsilon}{\Longrightarrow} p' \text{ and } q \overset{\epsilon}{\Longrightarrow} q' \text{ and } r = p' \| q' \tag{6}$$

   Using $p \xrightarrow{\tau^0} p'$ iff $p = p'$:

$$
\begin{array}{ll}
 & p \| q \overset{\epsilon}{\Longrightarrow} r \\
\text{implies} & (*\ \text{definition 2.2}\ *) \\
 & \exists n \geq 0 : p \| q \xrightarrow{\tau^n} r \\
\text{implies} & (*\ \text{definition } \|\ *) \\
 & \exists n \geq 0,\ \exists n_1, n_2 \geq 0 : n = n_1 + n_2 \\
 & \text{and } \exists p', q' : p \xrightarrow{\tau^{n_1}} p' \text{ and } q \xrightarrow{\tau^{n_2}} q' \text{ and } r = p' \| q'
\end{array}
$$

implies (∗ definition 2.2 ∗)
$\exists p', q' : \ p \stackrel{\epsilon}{\Longrightarrow} p' \ \text{ and } \ q \stackrel{\epsilon}{\Longrightarrow} q' \ \text{ and } \ r = p' \,\|\, q'$

$\sigma = a{\cdot}\sigma'$:

$p \,\|\, q \stackrel{a\cdot\sigma'}{\Longrightarrow} r$
implies (∗ definition 2.2 ∗)
$\exists r_1, r_2 : \ p \,\|\, q \stackrel{\epsilon}{\Longrightarrow} r_1 \stackrel{a}{\longrightarrow} r_2 \stackrel{\sigma'}{\Longrightarrow} r$
implies (∗ equation (6) ∗)
$\exists p_1, q_1, r_2 : \ p \stackrel{\epsilon}{\Longrightarrow} p_1 \text{ and } q \stackrel{\epsilon}{\Longrightarrow} q_1 \ \text{ and } \ p_1 \,\|\, q_1 \stackrel{a}{\longrightarrow} r_2 \stackrel{\sigma'}{\Longrightarrow} r$
implies (∗ definition $\|$ ∗)
$\exists p_1, q_1, p_2, q_2 : \ p \stackrel{\epsilon}{\Longrightarrow} p_1 \text{ and } q \stackrel{\epsilon}{\Longrightarrow} q_1$
$\text{ and } \ p_1 \stackrel{a}{\longrightarrow} p_2 \text{ and } q_1 \stackrel{a}{\longrightarrow} q_2 \ \text{ and } \ p_2 \,\|\, q_2 \stackrel{\sigma'}{\Longrightarrow} r$
implies (∗ induction ∗)
$\exists p_1, q_1, p_2, q_2, p', q' : \ p \stackrel{\epsilon}{\Longrightarrow} p_1 \text{ and } q \stackrel{\epsilon}{\Longrightarrow} q_1 \ \text{ and}$
$p_1 \stackrel{a}{\longrightarrow} p_2 \text{ and } q_1 \stackrel{a}{\longrightarrow} q_2 \ \text{ and } \ p_2 \stackrel{\sigma'}{\Longrightarrow} p' \ \text{ and } \ q_2 \stackrel{\sigma'}{\Longrightarrow} q' \ \text{ and } \ r = p' \,\|\, q'$
implies (∗ definition 2.2 ∗)
$\exists p', q' : \ p \stackrel{a\cdot\sigma'}{\Longrightarrow} p' \ \text{ and } \ q \stackrel{a\cdot\sigma'}{\Longrightarrow} q' \ \text{ and } \ r = p' \,\|\, q'$

2. By induction on the structure of σ:

$\sigma = \epsilon$: The lemma reduces to:

$$p \stackrel{\epsilon}{\Longrightarrow} p' \ \text{ and } \ q \stackrel{\epsilon}{\Longrightarrow} q' \quad \text{imply} \quad p \,\|\, q \stackrel{\epsilon}{\Longrightarrow} p' \,\|\, q' \tag{7}$$

which is straightforward:

$p \stackrel{\epsilon}{\Longrightarrow} q' \ \text{ and } \ q \stackrel{\epsilon}{\Longrightarrow} q'$
implies (∗ definition 2.2 ∗)
$\exists n_1, n_2 \geq 0 : \ p \stackrel{\tau^{n_1}}{\longrightarrow} q' \ \text{ and } \ q \stackrel{\tau^{n_2}}{\longrightarrow} q'$
implies (∗ definition $\|$ ∗)
$\exists n_1, n_2 \geq 0 : \ p \,\|\, q \stackrel{\tau^{n_1+n_2}}{\longrightarrow} p' \,\|\, q'$
implies (∗ definition 2.2 ∗)
$p \,\|\, q \stackrel{\epsilon}{\Longrightarrow} p' \,\|\, q'$

$\sigma = a{\cdot}\sigma'$:

$p \stackrel{a\cdot\sigma'}{\Longrightarrow} q' \ \text{ and } \ q \stackrel{a\cdot\sigma'}{\Longrightarrow} q'$
implies (∗ definition 2.2 ∗)
$\exists p_1, p_2, q_1, q_2 : \ p \stackrel{\epsilon}{\Longrightarrow} p_1 \stackrel{a}{\longrightarrow} p_2 \stackrel{\sigma'}{\Longrightarrow} p' \ \text{ and } \ q \stackrel{\epsilon}{\Longrightarrow} q_1 \stackrel{a}{\longrightarrow} q_2 \stackrel{\sigma'}{\Longrightarrow} q'$
implies (∗ equation (7), definition $\|$ and induction ∗)
$p \,\|\, q \stackrel{\epsilon}{\Longrightarrow} p_1 \,\|\, q_1 \stackrel{a}{\longrightarrow} p_2 \,\|\, q_2 \stackrel{\sigma'}{\Longrightarrow} p' \,\|\, q'$
implies (∗ definition 2.2 ∗)
$p \,\|\, q \stackrel{a\cdot\sigma'}{\Longrightarrow} p' \,\|\, q'$

□

Lemma A.2

1. Let $p, u \in \mathcal{LTS}(L)$, $\sigma \in L^*$, then

$$\sigma \in obs_c(u, p) \quad \text{iff} \quad \exists u' : \ u \stackrel{\sigma}{\Longrightarrow} u' \ \text{ and } \ p \ \textbf{after}\ \sigma\ \textbf{refuses}\ \{b \in L \mid u' \stackrel{b}{\Longrightarrow}\}$$

2. Let $\sigma \in L^*$ and $\sigma = b_1{\cdot}b_2{\cdot}\ldots{\cdot}b_m$, let $A \subseteq L$, and let $u_{[\sigma,A]} \in \mathcal{LTS}(L)$ be defined as:

$$u_{[\sigma,A]} \ =_{\text{def}} \ b_1 \ ; \ b_2 \ ; \ \ldots \ ; \ b_m \ ; \ \Sigma\{a; \textbf{stop} \mid a \in A\}$$

then

$$\sigma \in obs_c(u_{[\sigma,A]}, p) \quad \text{iff} \quad p \ \textbf{after}\ \sigma\ \textbf{refuses}\ A$$

□

Proof (lemma A.2)

1. $\quad\sigma \in obs_c(u,p)$

iff $\quad$ (∗ definition 3.2.3 ∗)

$\quad u \,\|\, p$ **after** σ **deadlocks**

iff $\quad$ (∗ definitions 3.2.2 and 3.2.1, lemmata A.1.1 and A.1.2 ∗)

$\quad \exists u', p' : \; u \stackrel{\sigma}{\Longrightarrow} u' \;$ and $\; p \stackrel{\sigma}{\Longrightarrow} p' \;$ and $\; \forall a \in L : \; u' \,\|\, p' \stackrel{a}{\not\Longrightarrow}$

iff $\quad$ (∗ lemmata A.1.2 and A.1.1 ∗)

$\quad \exists u', p' : \; u \stackrel{\sigma}{\Longrightarrow} u' \;$ and $\; p \stackrel{\sigma}{\Longrightarrow} p' \;$ and $\; \forall a \in L : \; u' \stackrel{a}{\not\Longrightarrow} \;$ or $\; p' \stackrel{a}{\not\Longrightarrow}$

iff $\quad$ (∗ standard set theory ∗)

$\quad \exists u', p' : \; u \stackrel{\sigma}{\Longrightarrow} u' \;$ and $\; p \stackrel{\sigma}{\Longrightarrow} p' \;$ and $\; \forall a \in \{ b \in L \mid u' \stackrel{b}{\Longrightarrow} \} : \; p' \stackrel{a}{\not\Longrightarrow}$

iff $\quad$ (∗ definition 3.2.1 ∗)

$\quad \exists u' : \; u \stackrel{\sigma}{\Longrightarrow} u' \;$ and $\; p$ **after** σ **refuses** $\{ b \in L \mid u' \stackrel{b}{\Longrightarrow} \}$

2. Using lemma A.2.1 with
   - $u_{[\sigma,A]} \stackrel{\sigma}{\Longrightarrow} u' \;$ implies $\; u' = \Sigma\{a; \mathbf{stop} \mid a \in A\}$
   - $u_{[\sigma,A]} \stackrel{\sigma}{\Longrightarrow} \Sigma\{a; \mathbf{stop} \mid a \in A\}$
   - $\{ b \in L \mid \Sigma\{a; \mathbf{stop} \mid a \in A\} \stackrel{b}{\Longrightarrow} \} \; = \; A$

□

Lemma A.3
$(\; \forall u \in \mathcal{LTS}(L) : obs_c(u,i) \subseteq obs_c(u,s) \;) \quad$ implies $\quad (\; \forall u \in \mathcal{LTS}(L) : obs_t(u,i) \subseteq obs_t(u,s) \;)$

□

Proof (lemma A.3)
Let $u \in \mathcal{LTS}(L)$, $\; \sigma \in obs_t(u,i)$, then

$\quad \sigma \in obs_t(u,i)$

implies $\quad$ (∗ definition 3.2.3 ∗)

$\quad u \,\|\, i \stackrel{\sigma}{\Longrightarrow}$

implies $\quad$ (∗ lemma A.1.1 ∗)

$\quad \exists u', i' : \; i \stackrel{\sigma}{\Longrightarrow} i' \;$ and $\; u \stackrel{\sigma}{\Longrightarrow} u'$

Now, let $\sigma = b_1 \cdot b_2 \cdot \ldots \cdot b_m$ and define $u_\sigma \in \mathcal{LTS}(L)$ as $u_\sigma \; =_{\mathrm{def}} \; b_1; b_2; \ldots; b_m; \mathbf{stop}$, then

$\quad u_\sigma \stackrel{\sigma}{\Longrightarrow} \mathbf{stop} \;$ and $\; \exists i' : i \stackrel{\sigma}{\Longrightarrow} i'$

implies $\quad$ (∗ lemma A.1.2 and A.1.1 ∗)

$\quad \exists i' : \; u_\sigma \,\|\, i \stackrel{\sigma}{\Longrightarrow} \mathbf{stop} \,\|\, i' \;$ and $\; \forall a \in L : \mathbf{stop} \,\|\, i' \stackrel{a}{\not\Longrightarrow}$

implies $\quad$ (∗ definitions 3.2.2 and 3.2.1 ∗)

$\quad u_\sigma \,\|\, i$ **after** σ **deadlocks**

implies $\quad$ (∗ definition 3.2.3 ∗)

$\quad \sigma \in obs_c(u_\sigma, i)$

implies $\quad$ (∗ premiss ∗)

$\quad \sigma \in obs_c(u_\sigma, s)$

implies $\quad$ (∗ definition 3.2.3 ∗)

$\quad u_\sigma \,\|\, s$ **after** σ **deadlocks**

implies $\quad$ (∗ definitions 3.2.2 and 3.2.1 ∗)

$\quad u_\sigma \,\|\, s \stackrel{\sigma}{\Longrightarrow}$

implies $\quad$ (∗ lemma A.1.1 ∗)

$\quad s \stackrel{\sigma}{\Longrightarrow}$

implies $\quad$ (∗ $u \stackrel{\sigma}{\Longrightarrow} u'$ from above and lemma A.1.2 ∗)

$\quad u \,\|\, s \stackrel{\sigma}{\Longrightarrow}$

implies $\quad$ (∗ definition 3.2.3 ∗)

$\sigma \in obs_t(u, s)$ $\square$

Lemma A.4

$\forall u \in \mathcal{LTS}(L) : obs_c(u, i) \subseteq obs_c(u, s)$

iff $\quad \forall \sigma \in L^*, \forall A \subseteq L : \; i$ **after** σ **refuses** A implies s **after** σ **refuses** A $\square$

Proof (lemma A.4)

only if : Let $\sigma \in L^*$, $A \subseteq L$, then

i **after** σ **refuses** A

iff (∗ lemma A.2.2 ∗)

$\sigma \in obs_c(u_{[\sigma,A]}, i)$

implies (∗ premiss ∗)

$\sigma \in obs_c(u_{[\sigma,A]}, s)$

iff (∗ lemma A.2.2 ∗)

s **after** σ **refuses** A

if: Let $u \in \mathcal{LTS}(L)$, $\sigma \in L^*$, then

$\sigma \in obs_c(u, i)$

iff (∗ lemma A.2.1 ∗)

$\exists u' : u \stackrel{\sigma}{\Longrightarrow} u'$ and i **after** σ **refuses** $\{b \in L \mid u' \stackrel{b}{\Longrightarrow}\}$

implies (∗ premiss ∗)

$\exists u' : u \stackrel{\sigma}{\Longrightarrow} u'$ and s **after** σ **refuses** $\{b \in L \mid u' \stackrel{b}{\Longrightarrow}\}$

iff (∗ lemma A.2.1 ∗)

$\sigma \in obs_c(u, s)$ $\square$

Proposition 3.3

$i \leq_{te} s$ iff $\forall \sigma \in L^*, \forall A \subseteq L : \; i$ **after** σ **refuses** A implies s **after** σ **refuses** A $\square$

Proof (proposition 3.3)

$i \leq_{te} s$

iff (∗ definition 3.2.4 ∗)

$\forall u \in \mathcal{LTS}(L) : \; obs_c(u, i) \subseteq obs_c(u, s)$ and $obs_t(u, i) \subseteq obs_t(u, s)$

iff (∗ lemma A.3 ∗)

$\forall u \in \mathcal{LTS}(L) : \; obs_c(u, i) \subseteq obs_c(u, s)$

iff (∗ lemma A.4 ∗)

$\forall \sigma \in L^*, \forall A \subseteq L : \; i$ **after** σ **refuses** A implies s **after** σ **refuses** A $\square$

Proposition 3.5

i **conf** s iff $\forall \sigma \in traces(s), \forall A \subseteq L : \; i$ **after** σ **refuses** A implies s **after** σ **refuses** A $\square$

Proof (proposition 3.5)

only if: Let $\sigma \in traces(s)$, $A \subseteq L$, then

$\sigma \in traces(s)$ and i **after** σ **refuses** A

iff (∗ lemma A.2.2 ∗)

$\sigma \in traces(s)$ and $\sigma \in obs_c(u_{[\sigma,A]}, i)$

implies (∗ premiss, definition 3.4 ∗)

$\sigma \in obs_c(u_{[\sigma,A]}, s)$

iff (∗ lemma A.2.2 ∗)

s **after** σ **refuses** A

if: Let $u \in \mathcal{LTS}(L)$, $\sigma \in L^*$, then

	$\sigma \in obs_c(u,i) \;\cap\; traces(s)$
iff	$(*$ lemma A.2.1 $*)$
	$\exists u' :\; u \xRightarrow{\sigma} u'$ and i **after** σ **refuses** $\{b \in L \mid u' \xRightarrow{b}\}$ and $\sigma \in traces(s)$
implies	$(*$ premiss $*)$
	$\exists u' :\; u \xRightarrow{\sigma} u'$ and s **after** σ **refuses** $\{b \in L \mid u' \xRightarrow{b}\}$
iff	$(*$ lemma A.2.1 $*)$
	$\sigma \in obs_c(u,s)$

Moreover

	$\sigma \in obs_t(u,i) \;\cap\; traces(s)$
implies	$(*$ definitions 3.2.3 and 2.3 and lemma A.1.1 $*)$
	$u \xRightarrow{\sigma}$ and $i \xRightarrow{\sigma}$ and $s \xRightarrow{\sigma}$
implies	$(*$ lemma A.1.2 $*)$
	$u \,\|\, s \xRightarrow{\sigma}$
implies	$(*$ definition 3.2.3 $*)$
	$\sigma \in obs_t(u,s)$

□

For the proof of proposition 3.8, we first introduce the notion of observing a refusal $A \subseteq L$. This is done by defining the transition relation $\xRightarrow{\varphi}\!\!\succ$ on observers in definition A.5, which is complementary to $\xRightarrow{\varphi}$ (definition 3.7). Whereas $p \xRightarrow{\varphi}$ expresses that φ is a failure trace of p, the transition relation $u \xRightarrow{\varphi}\!\!\succ$ expresses that the observer u can observe the failure trace φ in a system under observation. The failure trace φ is observed by performing the trace $\Theta(\varphi)$, which is φ in which all occurrences of refusal sets have been replaced by θ (definition A.6). The rest of the proof of proposition 3.8 is analogous to the proof of proposition 3.3: lemma A.8 investigates the behaviour of the parallel operator $\rceil\!|$, definition A.9 defines a special observer, for which it is shown in lemma A.10 that it can observe a particular failure trace. Using these special observers it is straightforward to prove that inclusion of failure-traces corresponds to inclusion of observations of the kind obs_c^θ (lemma A.11), as well as to inclusion of observations of the kind obs_t^θ (lemma A.12). Proposition 3.8 is then a direct consequence of these lemmata.

Definition A.5
Let $u \in \mathcal{LTS}(L_\theta)$, $a \in L$, $A \subseteq L$, $\alpha_{(i)} \in L \cup \mathcal{P}(L)$, and $\varphi \in (L \cup \mathcal{P}(L))^*$.

$u \xrightarrow{A}\!\!\succ u'$	$=_{\text{def}}$	$init(u) = A \cup \{\theta\}$ and $u \xrightarrow{\theta} u'$
$u \xRightarrow{\epsilon}\!\!\succ u'$	$=_{\text{def}}$	$u \xRightarrow{\epsilon} u'$
$u \xRightarrow{a}\!\!\succ u'$	$=_{\text{def}}$	$u \xRightarrow{a} u'$
$u \xRightarrow{A}\!\!\succ u'$	$=_{\text{def}}$	$\exists u_1, u_2 :\; u \xRightarrow{\epsilon}\!\!\succ u_1 \xrightarrow{A}\!\!\succ u_2 \xRightarrow{\epsilon}\!\!\succ u'$
$u \xRightarrow{\alpha_1\cdot\alpha_2\cdot\ldots\cdot\alpha_n}\!\!\succ u'$	$=_{\text{def}}$	$\exists u_0 \ldots u_n :\; u = u_0 \xRightarrow{\alpha_1}\!\!\succ u_1 \xRightarrow{\alpha_2}\!\!\succ \ldots \xRightarrow{a_n}\!\!\succ u_n = u'$
$u \xRightarrow{\varphi}\!\!\succ$	$=_{\text{def}}$	$\exists u' :\; u \xRightarrow{\varphi}\!\!\succ u'$
$u \stackrel{\varphi}{\not\Rightarrow}\!\!\succ$	$=_{\text{def}}$	not $\exists u' : u \xRightarrow{\varphi}\!\!\succ u'$

□

Definition A.6
Let $\varphi \in (L \cup \mathcal{P}(L))^*$, then $\Theta(\varphi) \in L_\theta^*$ is defined by ($a \in L$, $A \subseteq L$):

$$
\begin{array}{lll}
\Theta(\epsilon) & =_{\text{def}} & \epsilon \\
\Theta(a\cdot\varphi) & =_{\text{def}} & a\cdot\Theta(\varphi) \\
\Theta(A\cdot\varphi) & =_{\text{def}} & \theta\cdot\Theta(\varphi)
\end{array}
$$

□

Lemma A.7
Let $u \in \mathcal{LTS}(L_\theta)$, $\varphi \in (L \cup \mathcal{P}(L))^*$, then $\;u \overset{\varphi}{\Longrightarrow\!\!\!\!\succ} u'\;$ implies $\;u \xRightarrow{\Theta(\varphi)} u'$

□

Proof (lemma A.7)
By induction of the length of φ:

$\varphi = \epsilon$:
$u \overset{\epsilon}{\Longrightarrow\!\!\!\!\succ} u'$ iff $u \overset{\epsilon}{\Longrightarrow} u'$, and $\Theta(\epsilon) = \epsilon$

$\varphi = a \cdot \varphi'$:

$$
\begin{array}{ll}
 & u \overset{a\cdot\varphi'}{\Longrightarrow\!\!\!\!\succ} u' \\
\text{implies} & \exists u_1 : \; u \overset{a}{\Longrightarrow\!\!\!\!\succ} u_1 \;\text{ and }\; u_1 \overset{\varphi'}{\Longrightarrow\!\!\!\!\succ} u' \\
\text{implies} & (*\;\text{ definition A.5 and induction }\; *) \\
 & \exists u_1 : \; u \overset{a}{\Longrightarrow} u_1 \;\text{ and }\; u_1 \xRightarrow{\Theta(\varphi')} u' \\
\text{implies} & (*\;\text{ definition A.6 }\; *) \\
 & u \xRightarrow{\Theta(a\cdot\varphi')} u'
\end{array}
$$

$\varphi = A \cdot \varphi'$:

$$
\begin{array}{ll}
 & u \overset{A\cdot\varphi'}{\Longrightarrow\!\!\!\!\succ} u' \\
\text{implies} & \exists u_1, u_2 : \; u \overset{\epsilon}{\Longrightarrow\!\!\!\!\succ} u_1 \overset{A}{\longrightarrow\!\!\!\!\succ} u_2 \overset{\varphi'}{\Longrightarrow\!\!\!\!\succ} u' \\
\text{implies} & (*\;\text{ definitions A.5 and 2.3, and induction }\; *) \\
 & \exists u_1, u_2 : \; u \overset{\epsilon}{\Longrightarrow} u_1 \xrightarrow{\theta} u_2 \xRightarrow{\Theta(\varphi')} u' \\
\text{implies} & (*\;\text{ definition A.6 }\; *) \\
 & u \xRightarrow{\Theta(A\cdot\varphi')} u'
\end{array}
$$

□

Lemma A.8
Let $u, r \in \mathcal{LTS}(L_\theta)$, $\;p \in \mathcal{LTS}(L)$, $\;\sigma \in L_\theta^*$, and $\varphi \in (L \cup \mathcal{P}(L))^*$, then

1. $u \rceil| \, p \overset{\sigma}{\Longrightarrow} r \quad$ implies $\quad \exists u', p', \; \exists \varphi \in (L \cup \mathcal{P}(L))^* :$
   $u \overset{\varphi}{\Longrightarrow\!\!\!\!\succ} u'$ and $p \overset{\varphi}{\Longrightarrow} p'$ and $r = u' \rceil| \, p'$ and $\Theta(\varphi) = \sigma$

2. $u \overset{\varphi}{\Longrightarrow\!\!\!\!\succ} u'$ and $p \overset{\varphi}{\Longrightarrow} p'$ imply $u \rceil| \, p \xRightarrow{\Theta(\varphi)} u' \rceil| \, p'$

□

Proof (lemma A.8)

1. By induction on the structure of σ, with $a \in L$:

   $\sigma = \epsilon$: Using definition A.6, the lemma reduces to

   $$u \rceil| \, p \overset{\epsilon}{\Longrightarrow} r \;\text{ implies }\; \exists u', p' : \; u \overset{\epsilon}{\Longrightarrow\!\!\!\!\succ} u' \text{ and } p \overset{\epsilon}{\Longrightarrow} p' \text{ and } r = u' \rceil| \, p' \tag{8}$$

   which, using $p \xrightarrow{\tau^0} p'$ iff $p = p'$, is equivalent to

   $$u \rceil| \, p \xrightarrow{\tau^n} r \;\text{ implies }\; \exists u', p' : \; u \overset{\epsilon}{\Longrightarrow\!\!\!\!\succ} u' \text{ and } p \overset{\epsilon}{\Longrightarrow} p' \text{ and } r = u' \rceil| \, p' \tag{9}$$

   which is proved by induction on n:

   $n = 0$: Take $u' = u$, $p' = p$, then (9) is fulfilled.

   $n = n' + 1$:

   $$
   \begin{array}{ll}
    & u \rceil| \, p \xrightarrow{\tau^{(n'+1)}} r \\
   \text{implies} & (*\;\text{ definition 2.2 }\; *) \\
    & \exists r_1 : \; u \rceil| \, p \xrightarrow{\tau} r_1 \text{ and } r_1 \xrightarrow{\tau^{n'}} r
   \end{array}
   $$

implies (∗ definition 3.6.1 ∗)

$\exists r_1 :\ (\ \exists u_1 : u \xrightarrow{\tau} u_1$ and $r_1 = u_1\rceil\!|\,p$ and $r_1 \xrightarrow{\tau^{n'}} r\ )$ or

$(\ \exists p_1 : p \xrightarrow{\tau} p_1$ and $r_1 = u\rceil\!|\,p_1$ and $r_1 \xrightarrow{\tau^{n'}} r\ )$

implies (∗ induction ∗)

$(\ \exists u_1, u', p' :\ u \xrightarrow{\tau} u_1 \overset{\epsilon}{\Longrightarrow\!\!\!\succ} u'$ and $p \overset{\epsilon}{\Longrightarrow} p'$ and $r = u'\rceil\!|\,p'\ )$ or

$(\ \exists p_1, u', p' :\ u \overset{\epsilon}{\Longrightarrow\!\!\!\succ} u'$ and $p \xrightarrow{\tau} p_1 \overset{\epsilon}{\Longrightarrow} p'$ and $r = u'\rceil\!|\,p'\ )$

implies (∗ definition A.5 ∗)

$\exists u', p' :\ u \overset{\epsilon}{\Longrightarrow\!\!\!\succ} u'$ and $p \overset{\epsilon}{\Longrightarrow} p'$ and $r = u'\rceil\!|\,p'$

$\sigma = a{\cdot}\sigma'$:

$u\rceil\!|\,p \overset{a\cdot\sigma'}{\Longrightarrow} r$

implies (∗ definition 2.2 ∗)

$\exists r_1, r_2 :\ u\rceil\!|\,p \overset{\epsilon}{\Longrightarrow} r_1 \xrightarrow{a} r_2 \overset{\sigma'}{\Longrightarrow} r$

implies (∗ equation (8) ∗)

$\exists u_1, p_1, r_2 :\ u \overset{\epsilon}{\Longrightarrow\!\!\!\succ} u_1$ and $p \overset{\epsilon}{\Longrightarrow} p_1$ and $u_1\rceil\!|\,p_1 \xrightarrow{a} r_2 \overset{\sigma'}{\Longrightarrow} r$

implies (∗ definition 3.6.1 ∗)

$\exists u_1, p_1, u_2, p_2 :\ u \overset{\epsilon}{\Longrightarrow\!\!\!\succ} u_1$ and $p \overset{\epsilon}{\Longrightarrow} p_1$

and $u_1 \xrightarrow{a} u_2$ and $p_1 \xrightarrow{a} p_2$ and $u_2\rceil\!|\,p_2 \overset{\sigma'}{\Longrightarrow} r$

implies (∗ definition A.5, induction ∗)

$\exists u_2, p_2, u', p',\ \exists \varphi' \in (L \cup \mathcal{P}(L))^* :\ u \overset{a}{\Longrightarrow\!\!\!\succ} u_2$ and $p \overset{a}{\Longrightarrow} p_2$

and $u_2 \overset{\varphi'}{\Longrightarrow\!\!\!\succ} u'$ and $p_2 \overset{\varphi'}{\Longrightarrow} p'$ and $r = u'\rceil\!|\,p'$ and $\Theta(\varphi') = \sigma'$

implies (∗ definitions A.5, A.6; take $\varphi = a{\cdot}\varphi'$ ∗)

$\exists u', p',\ \exists \varphi \in (L \cup \mathcal{P}(L))^* :$

$u \overset{\varphi}{\Longrightarrow\!\!\!\succ} u'$ and $p \overset{\varphi}{\Longrightarrow} p'$ and $r = u'\rceil\!|\,p'$ and $\Theta(\varphi) = a{\cdot}\sigma'$

$\sigma = \theta{\cdot}\sigma'$:

$u\rceil\!|\,p \overset{\theta\cdot\sigma'}{\Longrightarrow} r$

implies (∗ definition 2.2 ∗)

$\exists r_1, r_2 :\ u\rceil\!|\,p \overset{\epsilon}{\Longrightarrow} r_1 \xrightarrow{\theta} r_2 \overset{\sigma'}{\Longrightarrow} r$

implies (∗ equation (8) ∗)

$\exists u_1, p_1, r_2 :\ u \overset{\epsilon}{\Longrightarrow\!\!\!\succ} u_1$ and $p \overset{\epsilon}{\Longrightarrow} p_1$ and $u_1\rceil\!|\,p_1 \xrightarrow{\theta} r_2 \overset{\sigma'}{\Longrightarrow} r$

implies (∗ definition 3.6.1 ∗)

$\exists u_1, p_1, u_2 :\ u \overset{\epsilon}{\Longrightarrow\!\!\!\succ} u_1$ and $p \overset{\epsilon}{\Longrightarrow} p_1$ and $u_1 \xrightarrow{\theta} u_2$ and

$u_1 \overset{\tau}{\not\longrightarrow}$ and $p_1 \overset{\tau}{\not\longrightarrow}$ and $\forall a \in L : u_1 \overset{a}{\not\longrightarrow}$ or $p_1 \overset{a}{\not\longrightarrow}$ and $u_2\rceil\!|\,p_1 \overset{\sigma'}{\Longrightarrow} r$

implies (∗ definitions 3.7, A.5; induction ∗)

$\exists u_1, p_1, u_2, u', p',\ \exists \varphi' \in (L \cup \mathcal{P}(L))^* :\ u \overset{\epsilon}{\Longrightarrow\!\!\!\succ} u_1$ and $p \overset{\epsilon}{\Longrightarrow} p_1$

and $u_1 \xrightarrow{init(u_1)\backslash\{\theta\}}\!\!\!\succ u_2$ and $p_1 \xrightarrow{init(u_1)\backslash\{\theta\}} p_1$

and $u_2 \overset{\varphi'}{\Longrightarrow\!\!\!\succ} u'$ and $p_1 \overset{\varphi'}{\Longrightarrow} p'$ and $r = u'\rceil\!|\,p'$ and $\Theta(\varphi') = \sigma'$

implies (∗ definitions A.5, A.6; take $\varphi = (init(u_1)\backslash\{\theta\}){\cdot}\varphi'$ ∗)

$\exists u', p',\ \exists \varphi \in (L \cup \mathcal{P}(L))^* :$

$u \overset{\varphi}{\Longrightarrow\!\!\!\succ} u'$ and $p \overset{\varphi}{\Longrightarrow} p'$ and $r = u'\rceil\!|\,p'$ and $\Theta(\varphi) = \theta{\cdot}\sigma'$

2. By induction on the structure of φ, with $a \in L$, $A \subseteq L$:

$\varphi = \epsilon$:

$u \overset{\epsilon}{\Longrightarrow\!\!\!\succ} u'$ and $p \overset{\epsilon}{\Longrightarrow} p'$

implies (∗ definition A.5 ∗)

$u \overset{\epsilon}{\Longrightarrow} u'$ and $p \overset{\epsilon}{\Longrightarrow} p'$

implies (∗ definition 2.2 ∗)

$\exists n, m :\ u \xrightarrow{\tau^n} u'$ and $p \xrightarrow{\tau^m} p'$

implies (∗ definition 3.6.1 ∗)

$\exists n, m: \ u \rceil| p \xrightarrow{\tau^{n+m}} u' \rceil| p'$

implies (∗ definition 2.2 ∗)

$u \rceil| p \xRightarrow{\epsilon} u' \rceil| p'$

implies (∗ definition A.6 ∗)

$u \rceil| p \xRightarrow{\Theta(\epsilon)} u' \rceil| p'$

$\varphi = a\cdot\varphi'$:

$u \xRightarrow{a\cdot\varphi'}\!\!\!\succ u'$ and $p \xRightarrow{a\cdot\varphi'} p'$

implies (∗ definitions 2.2, A.5 ∗)

$\exists u_1, u_2, p_1, p_2: \ u \xRightarrow{\epsilon} u_1 \xrightarrow{a} u_2 \xRightarrow{\varphi'}\!\!\!\succ u'$ and $p \xRightarrow{\epsilon} p_1 \xrightarrow{a} p_2 \xRightarrow{\varphi'} p'$

implies (∗ case for $\sigma = \epsilon$, definition 3.6.1, induction ∗)

$u \rceil| p \xRightarrow{\epsilon} u_1 \rceil| p_1 \xrightarrow{a} u_2 \rceil| p_2 \xRightarrow{\Theta(\varphi')} u' \rceil| p'$

implies (∗ definition A.6 ∗)

$u \rceil| p \xRightarrow{\Theta(a\cdot\varphi')} u' \rceil| p'$

$\varphi = A\cdot\varphi'$:

$u \xRightarrow{A\cdot\varphi'}\!\!\!\succ u'$ and $p \xRightarrow{A\cdot\varphi'} p'$

implies (∗ definitions 2.2, A.5 ∗)

$\exists u_1, u_2, p_1, p_2: \ u \xRightarrow{\epsilon} u_1 \overset{A}{\twoheadrightarrow} u_2 \xRightarrow{\varphi'}\!\!\!\succ u'$ and $p \xRightarrow{\epsilon} p_1 \xrightarrow{A} p_2 \xRightarrow{\varphi'} p'$

implies (∗ case for $\sigma = \epsilon$, definitions 3.7 and A.5 ∗)

$\exists u_1, u_2, p_1, p_2: \ u \rceil| p \xRightarrow{\epsilon} u_1 \rceil| p_1$ and $u_1 \xrightarrow{\theta} u_2$ and $\mathit{init}(u_1) = A \cup \{\theta\}$

and $p_1 = p_2$ and $\forall \mu \in A \cup \{\tau\}: p_1 \stackrel{\mu}{\not\longrightarrow}$ and $u_2 \xRightarrow{\varphi'}\!\!\!\succ u'$ and $p_2 \xRightarrow{\varphi'} p'$

implies (∗ definition 3.6.1, induction ∗)

$\exists u_1, u_2, p_1: \ u \rceil| p \xRightarrow{\epsilon} u_1 \rceil| p_1 \xrightarrow{\theta} u_2 \rceil| p_1 \xRightarrow{\Theta(\varphi')} u' \rceil| p'$

implies (∗ definition A.6 ∗)

$u \rceil| p \xRightarrow{\Theta(A\cdot\varphi')} u' \rceil| p'$

□

Definition A.9
Let $\varphi \in (L \cup \mathcal{P}(L))^*$ be a failure trace, then $u_{[\varphi]} \in \mathcal{LTS}(L_\theta)$ is defined as follows, where $a \in L$, and $A \subseteq L$:

$$\begin{array}{lll} u_{[\epsilon]} & =_{\text{def}} & \mathbf{stop} \\ u_{[a\cdot\varphi]} & =_{\text{def}} & a \ ; \ u_{[\varphi]} \\ u_{[A\cdot\varphi]} & =_{\text{def}} & \Sigma\{a; \mathbf{stop} \mid a \in A\} \ \Box \ \theta; u_{[\varphi]} \end{array}$$

□

Lemma A.10
Let $p \in \mathcal{LTS}(L)$, $\varphi, \psi \in (L \cup \mathcal{P}(L))^*$.

1. $u_{[\varphi]} \xRightarrow{\varphi}\!\!\!\succ \mathbf{stop}$
2. $u_{[\varphi]} \xRightarrow{\psi}\!\!\!\succ$ and $\Theta(\varphi) = \Theta(\psi)$ implies $\varphi = \psi$
3. $\varphi \in \mathit{Ftraces}(p)$ iff $\Theta(\varphi) \in \mathit{obs}_c^\theta(u_{[\varphi]}, p)$ iff $\Theta(\varphi) \in \mathit{obs}_t^\theta(u_{[\varphi]}, p)$

□

Proof (lemma A.10)

1. By induction on the structure of φ, with $a \in L$, $A \subseteq L$:

   $\varphi = \epsilon$: $u_{[\epsilon]} = \mathbf{stop} \xRightarrow{\epsilon}\!\!\!\succ \mathbf{stop}$

   $\varphi = a\cdot\varphi'$: $u_{[a\cdot\varphi']} = a; u_{[\varphi']} \xrightarrow{a} u_{[\varphi']}$, and by induction $u_{[\varphi']} \xRightarrow{\varphi'}\!\!\!\succ \mathbf{stop}$, so $u_{[a\cdot\varphi']} \xRightarrow{a\cdot\varphi'}\!\!\!\succ \mathbf{stop}$.

$\varphi = A{\cdot}\varphi'$: $u_{[A\cdot\varphi']} = \Sigma\{a;\mathbf{stop} \mid a \in A\} \mathbin{\Box} \theta; u_{[\varphi']} \xrightarrow{A}\!\!\!\!\succ u_{[\varphi']}$, and by induction:

$$u_{[\varphi']} \overset{\varphi'}{\Longrightarrow}\!\!\!\!\succ \mathbf{stop}, \text{ so } u_{[A\cdot\varphi']} \overset{A\cdot\varphi'}{\Longrightarrow}\!\!\!\!\succ \mathbf{stop}.$$

2. By induction on the structure of ψ, with $a \in L$, $A \subseteq L$:

$\psi = \epsilon$: $\Theta(\psi) = \Theta(\epsilon) = \epsilon$, so $\Theta(\varphi) = \epsilon$, hence $\varphi = \epsilon$.

$\psi = a{\cdot}\psi'$:

	$u_{[\varphi]} \overset{a\cdot\psi'}{\Longrightarrow}\!\!\!\!\succ$ and $\Theta(\varphi) = \Theta(a{\cdot}\psi')$
implies	(* definition A.6: $\Theta(a{\cdot}\psi') = a{\cdot}\Theta(\psi') = \Theta(\varphi)$, hence $\exists\varphi' : \varphi = a{\cdot}\varphi'$ so that $\Theta(\varphi) = a{\cdot}\Theta(\varphi')$ and $\Theta(\varphi') = \Theta(\psi')$ *)
	$\exists\varphi' : \varphi = a{\cdot}\varphi'$ and $u_{[a\cdot\varphi']} \overset{a\cdot\psi'}{\Longrightarrow}\!\!\!\!\succ$ and $\Theta(\varphi') = \Theta(\psi')$
implies	(* definition A.9 *)
	$\exists\varphi' : \varphi = a{\cdot}\varphi'$ and $u_{[\varphi']} \overset{\psi'}{\Longrightarrow}\!\!\!\!\succ$ and $\Theta(\varphi') = \Theta(\psi')$
implies	(* induction *)
	$\exists\varphi' : \varphi = a{\cdot}\varphi'$ and $\varphi' = \psi'$
implies	$\varphi = \psi$

$\psi = A{\cdot}\psi'$:

	$u_{[\varphi]} \overset{A\cdot\psi'}{\Longrightarrow}\!\!\!\!\succ$ and $\Theta(\varphi) = \Theta(A{\cdot}\psi')$
implies	(* definitions A.6 and A.9 *)
	$\exists\varphi' : \varphi = A{\cdot}\varphi'$ and $u_{[\varphi']} \overset{\psi'}{\Longrightarrow}\!\!\!\!\succ$ and $\Theta(\varphi') = \Theta(\psi')$
implies	(* induction *)
	$\exists\varphi' : \varphi = A{\cdot}\varphi'$ and $\varphi' = \psi'$
implies	$\varphi = \psi$

3. ◦ $\varphi \in \mathit{Ftraces}(p)$ implies $\Theta(\varphi) \in \mathit{obs}_c^\theta(u_{[\varphi]}, p)$:

	$\varphi \in \mathit{Ftraces}(p)$
implies	(* definition 3.7 and lemma A.10.1 *)
	$\exists p' : p \overset{\varphi}{\Longrightarrow} p'$ and $u_{[\varphi]} \overset{\varphi}{\Longrightarrow}\!\!\!\!\succ \mathbf{stop}$
implies	(* lemma A.8.2 and definition 3.6.1 *)
	$\exists p' : u_{[\varphi]} \rceil\!\vert\, p \overset{\Theta(\varphi)}{\Longrightarrow} \mathbf{stop} \rceil\!\vert\, p'$ and $\forall a \in L_\theta : \mathbf{stop} \rceil\!\vert\, p' \overset{a}{\not\Longrightarrow}$
implies	(* definition 3.2.2 *)
	$(u_{[\varphi]} \rceil\!\vert\, p)$ **after** $\Theta(\varphi)$ **deadlocks**
implies	(* definition 3.6.2 *)
	$\Theta(\varphi) \in \mathit{obs}_c^\theta(u_{[\varphi]}, p)$

◦ $\Theta(\varphi) \in \mathit{obs}_c^\theta(u_{[\varphi]}, p)$ implies $\Theta(\varphi) \in \mathit{obs}_t^\theta(u_{[\varphi]}, p)$:

	$\Theta(\varphi) \in \mathit{obs}_c^\theta(u_{[\varphi]}, p)$
implies	(* definitions 3.6.2, 3.2.2 *)
	$\exists r : u_{[\varphi]} \rceil\!\vert\, p \overset{\Theta(\varphi)}{\Longrightarrow} r$ and $\forall a \in L_\theta : r \overset{a}{\not\Longrightarrow}$
implies	(* definition 2.2 *)
	$u_{[\varphi]} \rceil\!\vert\, p \overset{\Theta(\varphi)}{\Longrightarrow}$
implies	(* definition 3.6.2 *)
	$\Theta(\varphi) \in \mathit{obs}_t^\theta(u_{[\varphi]}, p)$

◦ $\Theta(\varphi) \in \mathit{obs}_t^\theta(u_{[\varphi]}, p)$ implies $\varphi \in \mathit{Ftraces}(p)$:

	$\Theta(\varphi) \in \mathit{obs}_t^\theta(u_{[\varphi]}, p)$
implies	(* definition 3.6.2, lemma A.8.1 *)
	$\exists\psi \in (L \cup \mathcal{P}(L))^* : u_{[\varphi]} \overset{\psi}{\Longrightarrow}\!\!\!\!\succ$ and $p \overset{\psi}{\Longrightarrow}$ and $\Theta(\psi) = \Theta(\varphi)$
implies	(* lemma A.10.2 *)
	$p \overset{\varphi}{\Longrightarrow}$

implies (∗ definition 3.7 ∗)

$\varphi \in Ftraces(p)$

□

Lemma A.11

Let $i, s \in \mathcal{LTS}(L)$, then

$$Ftraces(i) \subseteq Ftraces(s) \quad \text{iff} \quad \forall u \in \mathcal{LTS}(L_\theta): \ obs_c^\theta(u,i) \subseteq obs_c^\theta(u,s)$$

□

Proof (lemma A.11)

only if: Let $u \in \mathcal{LTS}(L_\theta)$, $\sigma \in L_\theta^*$, then

$\sigma \in obs_c^\theta(u,i)$

implies (∗ definitions 3.6.2 and 3.2.2 ∗)

$(u\rceil\!|\, i)$ **after** σ **refuses** L_θ

implies (∗ claim (2) ∗)

$\exists r: \ u\rceil\!|\, i \overset{\sigma}{\Longrightarrow} r$ and $\forall \mu \in L_\theta \cup \{\tau\} : r \overset{\mu}{\not\longrightarrow}$

implies (∗ lemma A.8.1 ∗)

$\exists u', i', \ \exists \varphi \in (L \cup \mathcal{P}(L))^* : \ u \overset{\varphi}{\Longrightarrow\!\!\succ} u'$ and $i \overset{\varphi}{\Longrightarrow} i'$

and $\forall \mu \in L_\theta \cup \{\tau\} : u'\rceil\!|\, i' \overset{\mu}{\not\longrightarrow}$ and $\Theta(\varphi) = \sigma$

implies (∗ definition 3.6.1 ∗)

$\exists u', i', \ \exists \varphi \in (L \cup \mathcal{P}(L))^* : \ u \overset{\varphi}{\Longrightarrow\!\!\succ} u'$ and $i \overset{\varphi}{\Longrightarrow} i'$

and $u' \overset{\theta}{\not\longrightarrow}$ and $u' \overset{\tau}{\not\longrightarrow}$ and $i' \overset{\tau}{\not\longrightarrow}$ and $\forall a \in L : u' \overset{a}{\not\longrightarrow}$ or $i' \overset{a}{\not\longrightarrow}$

and $\Theta(\varphi) = \sigma$

implies (∗ definition 3.7 ∗)

$\exists u', i', \ \exists \varphi \in (L \cup \mathcal{P}(L))^* : \ u \overset{\varphi}{\Longrightarrow\!\!\succ} u' \overset{\theta}{\not\longrightarrow}$ and $u' \overset{\tau}{\not\longrightarrow}$

and $i \overset{\varphi}{\Longrightarrow} i' \xrightarrow{init(u')} i'$ and $\Theta(\varphi) = \sigma$

implies (∗ definition 3.7 ∗)

$\exists u', \ \exists \varphi \in (L \cup \mathcal{P}(L))^* : \ u \overset{\varphi}{\Longrightarrow\!\!\succ} u' \overset{\theta}{\not\longrightarrow}$ and $u' \overset{\tau}{\not\longrightarrow}$

and $\varphi \cdot init(u') \in Ftraces(i)$ and $\Theta(\varphi) = \sigma$

implies (∗ premiss ∗)

$\exists u', \ \exists \varphi \in (L \cup \mathcal{P}(L))^* : \ u \overset{\varphi}{\Longrightarrow\!\!\succ} u' \overset{\theta}{\not\longrightarrow}$ and $u' \overset{\tau}{\not\longrightarrow}$

and $\varphi \cdot init(u') \in Ftraces(s)$ and $\Theta(\varphi) = \sigma$

implies (∗ definition 3.7 ∗)

$\exists u', \ \exists \varphi \in (L \cup \mathcal{P}(L))^* : \ u \overset{\varphi}{\Longrightarrow\!\!\succ} u' \overset{\theta}{\not\longrightarrow}$ and $u' \overset{\tau}{\not\longrightarrow}$

and $\exists s' : \ s \overset{\varphi}{\Longrightarrow} s'$ and $\forall \mu \in init(u') \cup \{\tau\} : s' \overset{\mu}{\not\longrightarrow}$ and $\Theta(\varphi) = \sigma$

implies (∗ lemma A.8.2 and definition 3.6.1 ∗)

$\exists u', s', \ \exists \varphi \in (L \cup \mathcal{P}(L))^* : \ u\rceil\!|\, s \overset{\Theta(\varphi)}{\Longrightarrow} u'\rceil\!|\, s'$

and $\forall \mu \in L_\theta \cup \{\tau\} : u'\rceil\!|\, s' \overset{\mu}{\not\longrightarrow}$ and $\Theta(\varphi) = \sigma$

implies (∗ claim (2) ∗)

$\exists \varphi \in (L \cup \mathcal{P}(L))^* : \ (u\rceil\!|\, s)$ **after** $\Theta(\varphi)$ **refuses** L_θ and $\Theta(\varphi) = \sigma$

implies (∗ definitions 3.2.2 and 3.6.2 ∗)

$\sigma \in obs_c^\theta(u,s)$

if:

$\varphi \in Ftraces(i)$

implies (∗ lemma A.10.3 ∗)

$\Theta(\varphi) \in obs_c^\theta(u_{[\varphi]}, i)$

implies (∗ premiss ∗)

$\Theta(\varphi) \in obs_c^\theta(u_{[\varphi]}, s)$

implies (∗ lemma A.10.3 ∗)

$\varphi \in Ftraces(s)$

□

Lemma A.12
Let $i, s \in \mathcal{LTS}(L)$, then

$$Ftraces(i) \subseteq Ftraces(s) \quad \text{iff} \quad \forall u \in \mathcal{LTS}(L_\theta): \; obs_t^\theta(u,i) \subseteq obs_t^\theta(u,s)$$

$\Box$

Proof (lemma A.12)

only if: Let $u \in \mathcal{LTS}(L_\theta)$, $\sigma \in L_\theta^*$, then

$$\begin{array}{ll}
 & \sigma \in obs_t^\theta(u,i) \\
\text{implies} & (*\ \text{definition 3.6.2}\ *) \\
 & u \rceil\!| \, i \stackrel{\sigma}{\Longrightarrow} \\
\text{implies} & (*\ \text{lemma A.8.1}\ *) \\
 & \exists \varphi \in (L \cup \mathcal{P}(L))^* : \; u \stackrel{\varphi}{\Longrightarrow\!\!\!\succ} \text{ and } i \stackrel{\varphi}{\Longrightarrow} \text{ and } \Theta(\varphi) = \sigma \\
\text{implies} & (*\ \text{definition 3.7, premiss}\ *) \\
 & \exists \varphi \in (L \cup \mathcal{P}(L))^* : \; u \stackrel{\varphi}{\Longrightarrow\!\!\!\succ} \text{ and } s \stackrel{\varphi}{\Longrightarrow} \text{ and } \Theta(\varphi) = \sigma \\
\text{implies} & (*\ \text{lemma A.8.2}\ *) \\
 & \exists \varphi \in (L \cup \mathcal{P}(L))^* : \; u \rceil\!| \, s \xrightarrow{\Theta(\varphi)} \text{ and } \Theta(\varphi) = \sigma \\
\text{implies} & (*\ \text{definition 3.6.2}\ *) \\
 & \sigma \in obs_t^\theta(u,s)
\end{array}$$

if:

$$\begin{array}{ll}
 & \varphi \in Ftraces(i) \\
\text{implies} & (*\ \text{lemma A.10.3}\ *) \\
 & \Theta(\varphi) \in obs_t^\theta(u_{[\varphi]}, i) \\
\text{implies} & (*\ \text{premiss}\ *) \\
 & \Theta(\varphi) \in obs_t^\theta(u_{[\varphi]}, s) \\
\text{implies} & (*\ \text{lemma A.10.3}\ *) \\
 & \varphi \in Ftraces(s)
\end{array}$$

$\Box$

Proposition 3.8
$i \leq_{rf} s \quad \text{iff} \quad Ftraces(i) \subseteq Ftraces(s)$

$\Box$

Proof (proposition 3.8)

$$\begin{array}{ll}
 & i \leq_{rf} s \\
\text{iff} & (*\ \text{definition 3.6.3}\ *) \\
 & \forall u \in \mathcal{LTS}(L_\theta): \; obs_c^\theta(u,i) \subseteq obs_c^\theta(u,s) \text{ and } obs_t^\theta(u,i) \subseteq obs_t^\theta(u,s) \\
\text{iff} & (*\ \text{lemmata A.11 and A.12}\ *) \\
 & Ftraces(i) \subseteq Ftraces(s)
\end{array}$$

$\Box$

Proposition 3.9

1. $\leq_{tr}, \; \leq_{te}, \; \leq_{rf}$ are preorders; **conf** is reflexive, but not transitive.
2. $\leq_{rf} \; \subset \; \leq_{te} \; = \; \leq_{tr} \cap \, \mathbf{conf}$

$\Box$

Proof (proposition 3.9)

1. Reflexivity and transitivity of $\leq_{tr}$, $\leq_{te}$ and $\leq_{rf}$ follow directly from their respective definitions (definitions 3.1, 3.2.4, 3.6.3) using reflexivity and transitivity of $\subseteq$. Reflexivity of **conf** follows also directly from its definition (definition 3.4). Intransitivity of **conf** follows from, e.g., $p_1 = a; \mathbf{stop} \,\Box\, c; \mathbf{stop}$, $p_2 = c; \mathbf{stop}$, and $p_3 = a; b; \mathbf{stop} \,\Box\, \mathbf{i}; c; \mathbf{stop}$, then $p_1 \; \mathbf{conf} \; p_2$ and $p_2 \; \mathbf{conf} \; p_3$, but $p_1 \; \not\!\mathbf{conf} \; p_3$.

2. $\leq_{rf} \subseteq \leq_{te}$ follows from the fact that any failure pair can be expressed as a failure trace:

$$
\begin{array}{ll}
 & p \ \mathbf{after}\ \sigma \ \mathbf{refuses}\ A \\
\text{iff} & (*\ \text{claim (2)}\ *) \\
 & \exists p' :\ p \overset{\sigma}{\Longrightarrow} p' \ \text{ and }\ \forall \mu \in A \cup \{\tau\} :\ p' \overset{\mu}{\not\longrightarrow} \\
\text{iff} & (*\ \text{definition 3.7.1}\ *) \\
 & \exists p' :\ p \overset{\sigma}{\Longrightarrow} p' \overset{A}{\longrightarrow} p' \\
\text{iff} & (*\ \text{definition 3.7.2}\ *) \\
 & \sigma{\cdot}A \in \mathit{Ftraces}(p)
\end{array}
$$

$\leq_{rf} \neq \leq_{te}$ follows from figure 2 in section 3.

$\leq_{te} = \leq_{tr} \cap$ **conf** can be deduced from propositions 3.3 and 3.5, together with the following characterization of $\leq_{tr}$:

$$
\begin{array}{ll}
 & i \leq_{tr} s \\
\text{iff} & (*\ \text{definition 3.1}\ *) \\
 & \forall \sigma \notin \mathit{traces}(s) :\ \sigma \notin \mathit{traces}(i) \\
\text{iff} & (*\ \text{definition 3.2.1}\ *) \\
 & \forall \sigma \notin \mathit{traces}(s) :\ \forall A \subseteq L :\ \ \text{not}\ (\, i \ \mathbf{after}\ \sigma \ \mathbf{refuses}\ A\,) \\
\text{iff} & (*\ \text{standard logic;} \\
 & \quad \sigma \notin \mathit{traces}(s) \ \text{implies}\ \ \text{not}\ (\, s \ \mathbf{after}\ \sigma \ \mathbf{refuses}\ A\,)\ \text{for any}\ A \subseteq L\ \ *) \\
 & \forall \sigma \notin \mathit{traces}(s),\ \forall A \subseteq L :\ \ i \ \mathbf{after}\ \sigma \ \mathbf{refuses}\ A \ \text{ implies }\ s \ \mathbf{after}\ \sigma \ \mathbf{refuses}\ A
\end{array}
$$

□

A.2 Proofs of Section 4.1 (Implementation relations with inputs and outputs — Input-output testing relation)

Proposition 4.3
$i \leq_{iot} s \quad \text{iff} \quad traces(i) \subseteq traces(s) \;\text{ and }\; Qtraces(i) \subseteq Qtraces(s)$

$\square$

Proof (proposition 4.3)

only if: Let $\sigma \in traces(i)$, and define $u_\sigma \in \mathcal{IOTS}(L_U, L_I)$, such that $\exists u' : u_\sigma \xrightarrow{\sigma} u'$, then

$$
\begin{array}{ll}
 & i \stackrel{\sigma}{\Longrightarrow} \;\text{ and }\; u_\sigma \xrightarrow{\sigma} u' \\
\text{implies} & (*\ \text{lemma A.1.2}\ *) \\
 & u_\sigma \,\|\, i \stackrel{\sigma}{\Longrightarrow} \\
\text{implies} & (*\ \text{definition 3.2.2}\ *) \\
 & \sigma \in obs_t(u_\sigma, i) \\
\text{implies} & (*\ \text{premiss, definition 4.1}\ *) \\
 & \sigma \in obs_t(u_\sigma, s) \\
\text{implies} & (*\ \text{definition 3.2.2}\ *) \\
 & u_\sigma \,\|\, s \stackrel{\sigma}{\Longrightarrow} \\
\text{implies} & (*\ \text{lemma A.1.1}\ *) \\
 & s \stackrel{\sigma}{\Longrightarrow} \\
\text{implies} & (*\ \text{definition 2.3}\ *) \\
 & \sigma \in traces(s)
\end{array}
$$

Let $\sigma \in Qtraces(i)$, and define u_σ as above, with additionally $init(u') = L_U$, then

$$
\begin{array}{ll}
 & \sigma \in Qtraces(i) \;\text{ and }\; \exists u' : u_\sigma \xrightarrow{\sigma} u' \;\text{ and }\; init(u') = L_U \\
\text{implies} & (*\ \text{definition 4.2}\ *) \\
 & (\ \exists i' : i \stackrel{\sigma}{\Longrightarrow} i' \text{ and } \forall \mu \in L_U \cup \{\tau\} : i' \stackrel{\mu}{\not\longrightarrow}\ ) \;\text{ and} \\
 & (\ \exists u' : u_\sigma \stackrel{\sigma}{\Longrightarrow} u' \text{ and } \forall \mu \in L_I \cup \{\tau\} : u' \stackrel{\mu}{\not\longrightarrow}\ ) \\
\text{implies} & (*\ \text{lemma A.1.2 and definition } \|\ *) \\
 & \exists i', u' :\ u_\sigma \,\|\, i \stackrel{\sigma}{\Longrightarrow} u' \,\|\, i' \;\text{ and }\; \forall \mu \in L \cup \{\tau\} :\ u' \,\|\, i' \stackrel{\mu}{\not\longrightarrow} \\
\text{implies} & (*\ \text{claim (2) and definition 3.2.2}\ *) \\
 & u_\sigma \,\|\, i \;\textbf{after}\; \sigma \;\textbf{deadlocks} \\
\text{implies} & (*\ \text{definition 3.2.3}\ *) \\
 & \sigma \in obs_c(u_\sigma, i) \\
\text{implies} & (*\ \text{premiss, definition 4.1}\ *) \\
 & \sigma \in obs_c(u_\sigma, s) \\
\text{implies} & (*\ \text{definition 3.2.3}\ *) \\
 & u_\sigma \,\|\, s \;\textbf{after}\; \sigma \;\textbf{deadlocks} \\
\text{implies} & (*\ \text{definition 3.2.2 and claim (2)}\ *) \\
 & \exists r :\ u_\sigma \,\|\, s \stackrel{\sigma}{\Longrightarrow} r \;\text{ and }\; \forall \mu \in L \cup \{\tau\} :\ r \stackrel{\mu}{\not\longrightarrow} \\
\text{implies} & (*\ \text{lemma A.1.1}\ *) \\
 & \exists u', s' :\ u_\sigma \stackrel{\sigma}{\Longrightarrow} u' \;\text{ and }\; s \stackrel{\sigma}{\Longrightarrow} s' \;\text{ and }\; \forall \mu \in L \cup \{\tau\} :\ u' \,\|\, s' \stackrel{\mu}{\not\longrightarrow} \\
\text{implies} & (*\ \text{definition } \|\text{, and } u \in \mathcal{IOTS}(L_U, L_I)\text{, so } u' \stackrel{x}{\Longrightarrow} \text{ for all } x \in L_U\ *) \\
 & \exists s' : s \stackrel{\sigma}{\Longrightarrow} s' \text{ and } \forall x \in L_U \cup \{\tau\} :\ s' \stackrel{\mu}{\not\longrightarrow} \\
\text{implies} & (*\ \text{definition 4.2}\ *) \\
 & \sigma \in Qtraces(s)
\end{array}
$$

if: Let $u \in \mathcal{IOTS}(L_U, L_I)$, then

$$
\begin{array}{ll}
 & \sigma \in obs_c(u, i) \\
\text{implies} & (*\ \text{definition 3.2.3}\ *) \\
 & u \,\|\, i \;\textbf{after}\; \sigma \;\textbf{deadlocks}
\end{array}
$$

$$
\begin{array}{ll}
\text{implies} & (*\ \text{definition 3.2.2 and claim (2)}\ *) \\
 & \exists r:\ u \,\|\, i \stackrel{\sigma}{\Longrightarrow} r \ \text{ and }\ \forall \mu \in L \cup \{\tau\}:\ r \stackrel{\mu}{\not\longrightarrow} \\
\text{implies} & (*\ \text{lemma A.1.1}\ *) \\
 & \exists u', i':\ u \stackrel{\sigma}{\Longrightarrow} u' \text{ and } i \stackrel{\sigma}{\Longrightarrow} i' \text{ and } \forall \mu \in L \cup \{\tau\}:\ u' \,\|\, i' \stackrel{\mu}{\not\longrightarrow} \\
\text{implies} & (*\ \text{definition } \|\,; \\
 & \quad u' \in \mathcal{IOTS}(L_U, L_I), \text{ so } \forall x \in L_U : u' \stackrel{x}{\Longrightarrow}; \\
 & \quad i' \in \mathcal{IOTS}(L_I, L_U), \text{ so } \forall a \in L_I : i' \stackrel{a}{\Longrightarrow}\ \ *) \\
 & \exists u', i':\ u \stackrel{\sigma}{\Longrightarrow} u' \text{ and } i \stackrel{\sigma}{\Longrightarrow} i' \text{ and } \mathit{init}(u') = L_U \text{ and } \mathit{init}(i') = L_I \\
\text{implies} & (*\ \text{definition 4.2.1}\ *) \\
 & \exists u':\ u \stackrel{\sigma}{\Longrightarrow} u' \text{ and } \mathit{init}(u') = L_U \text{ and } \exists i':\ i \stackrel{\sigma}{\Longrightarrow} i' \text{ and } \delta(i') \\
\text{implies} & (*\ \text{definition 4.2.3}\ *) \\
 & \exists u':\ u \stackrel{\sigma}{\Longrightarrow} u' \text{ and } \mathit{init}(u') = L_U \text{ and } \sigma \in \mathit{Qtraces}(i) \\
\text{implies} & (*\ \text{premiss}\ *) \\
 & \exists u':\ u \stackrel{\sigma}{\Longrightarrow} u' \text{ and } \mathit{init}(u') = L_U \text{ and } \sigma \in \mathit{Qtraces}(s) \\
\text{implies} & (*\ \text{definition 4.2.3}\ *) \\
 & \exists u':\ u \stackrel{\sigma}{\Longrightarrow} u' \text{ and } \mathit{init}(u') = L_U \text{ and } \exists s':\ s \stackrel{\sigma}{\Longrightarrow} s' \text{ and } \delta(s') \\
\text{implies} & (*\ \text{definition 4.2.1}\ *) \\
 & \exists u':\ u \stackrel{\sigma}{\Longrightarrow} u' \text{ and } \mathit{init}(u') = L_U \text{ and} \\
 & \exists s':\ s \stackrel{\sigma}{\Longrightarrow} s' \text{ and } \forall \mu \in L_U \cup \{\tau\}:\ s' \stackrel{\mu}{\not\longrightarrow} \\
\text{implies} & (*\ \text{lemma A.1.2 and definition } \|\ \ *) \\
 & \exists u', s':\ u \,\|\, s \stackrel{\sigma}{\Longrightarrow} u' \,\|\, s' \text{ and } \forall \mu \in L \cup \{\tau\}:\ u' \,\|\, s' \stackrel{\mu}{\not\longrightarrow} \\
\text{implies} & (*\ \text{claim (2) and definition 3.2.2}\ *) \\
 & u \,\|\, s \ \textbf{after}\ \sigma\ \textbf{deadlocks} \\
\text{implies} & (*\ \text{definition 3.2.3}\ *) \\
 & \sigma \in \mathit{obs}_c(u, s)
\end{array}
$$

Let $u \in \mathcal{IOTS}(L_U, L_I)$, then

$$
\begin{array}{ll}
 & \sigma \in \mathit{obs}_t(u, i) \\
\text{implies} & (*\ \text{definition 3.2.3}\ *) \\
 & u \,\|\, i \stackrel{\sigma}{\Longrightarrow} \\
\text{implies} & (*\ \text{lemma A.1.1}\ *) \\
 & u \stackrel{\sigma}{\Longrightarrow} \ \text{ and }\ i \stackrel{\sigma}{\Longrightarrow} \\
\text{implies} & (*\ \text{definition 2.3}\ *) \\
 & u \stackrel{\sigma}{\Longrightarrow} \ \text{ and }\ \sigma \in \mathit{traces}(i) \\
\text{implies} & (*\ \text{premiss}\ *) \\
 & u \stackrel{\sigma}{\Longrightarrow} \ \text{ and }\ \sigma \in \mathit{traces}(s) \\
\text{implies} & (*\ \text{definition 2.3}\ *) \\
 & u \stackrel{\sigma}{\Longrightarrow} \ \text{ and }\ s \stackrel{\sigma}{\Longrightarrow} \\
\text{implies} & (*\ \text{lemma A.1.2}\ *) \\
 & u \,\|\, s \stackrel{\sigma}{\Longrightarrow} \\
\text{implies} & (*\ \text{definition 3.2.3}\ *) \\
 & \sigma \in \mathit{obs}_t(u, s)
\end{array}
$$

$\square$

Lemma A.13
Let $p \in \mathcal{LTS}(L)$, $\sigma \in L^*$, $x \in L_U$, then

1. $x \in \mathit{out}(\, p \ \textbf{after}\ \sigma\,)$ iff $\sigma \cdot x \in \mathit{traces}(p)$
2. $\delta \in \mathit{out}(\, p \ \textbf{after}\ \sigma\,)$ iff $\sigma \in \mathit{Qtraces}(p)$
3. $\mathit{out}(\, p \ \textbf{after}\ \sigma\,) \neq \emptyset$ iff $\sigma \in \mathit{traces}(p)$

$\square$

Proof (lemma A.13)

1. $x \in out(\, p\,\textbf{after}\,\sigma\,)$
   iff $(*\;$ definition 4.4.2 $\;*)$
   $\exists p' \in (\, p\,\textbf{after}\,\sigma\,) : x \in out(p')$
   iff $(*\;$ definitions 4.4.1 and 2.3 $\;*)$
   $\exists p' : p \overset{\sigma}{\Longrightarrow} p'$ and $p' \xrightarrow{x}$
   iff $(*\;$ definitions 2.2 and 2.3 $\;*)$
   $\sigma \cdot x \in traces(p)$

2. $\delta \in out(\, p\,\textbf{after}\,\sigma\,)$
   iff $(*\;$ definition 4.4.2 $\;*)$
   $\exists p' \in (\, p\,\textbf{after}\,\sigma\,) : \delta \in out(p')$
   iff $(*\;$ definition 4.4.1 $\;*)$
   $\exists p' \in (\, p\,\textbf{after}\,\sigma\,) : \delta(p')$
   iff $(*\;$ definition 4.2 $\;*)$
   $\sigma \in Qtraces(p)$

3. $out(\, p\,\textbf{after}\,\sigma\,) = \emptyset$
   iff $(*\;$ definition 4.4.2 $\;*)$
   $\forall p' \in (\, p\,\textbf{after}\,\sigma\,) : out(p') = \emptyset$
   iff $(*\;$ definitions 4.4.1 and 4.2 $\;*)$
   $\forall p' \in (\, p\,\textbf{after}\,\sigma\,) :\; \forall x \in L_U : p' \overset{x}{\not\longrightarrow}$ and $\exists \mu \in L_U \cup \{\tau\} : p' \xrightarrow{\mu}$
   iff $(*\;$ p is strongly convergent $\;*)$
   $p\,\textbf{after}\,\sigma\, = \emptyset$
   iff $(*\;$ definition 2.3 $\;*)$
   $\sigma \notin traces(p)$

$\square$

Proposition 4.5

$i \leq_{iot} s$ iff $\forall \sigma \in L^* :\; out(\, i\,\textbf{after}\,\sigma\,) \;\subseteq\; out(\, s\,\textbf{after}\,\sigma\,)$

$\square$

Proof (proposition 4.5)

only if: Let $\sigma \in L^*$ and $x \in L_U \cup \{\delta\}$, then

$x \in out(\, i\,\textbf{after}\,\sigma\,)$
implies $(*\;$ lemmata A.13.1. and A.13.2 $\;*)$
$(\; x \in L_U$ and $\sigma \cdot x \in traces(i)\;)$ or $(\; x = \delta$ and $\sigma \in Qtraces(i)\;)$
implies $(*\;$ premiss, proposition 4.3 $\;*)$
$(\; x \in L_U$ and $\sigma \cdot x \in traces(s)\;)$ or $(\; x = \delta$ and $\sigma \in Qtraces(s)\;)$
implies $(*\;$ lemmata A.13.1. and A.13.2 $\;*)$
$(\; x \in L_U$ and $x \in out(\, s\,\textbf{after}\,\sigma\,)\;)$ or $(\; x = \delta$ and $x \in out(\, s\,\textbf{after}\,\sigma\,)\;)$
implies $(*\;$ definition 4.4 $\;*)$
$x \in out(\, s\,\textbf{after}\,\sigma\,)$

if: Using proposition 4.3, let $\sigma \in traces(i)$, then

$\sigma \in traces(i)$
implies $(*\;$ lemma A.13.3 $\;*)$
$out(\, i\,\textbf{after}\,\sigma\,) \neq \emptyset$
implies $(*\;$ premiss $\;*)$
$out(\, s\,\textbf{after}\,\sigma\,) \neq \emptyset$
implies $(*\;$ lemma A.13.3 $\;*)$
$\sigma \in traces(s)$

Let $\sigma \in Qtraces(i)$, then

$$\begin{array}{ll} & \sigma \in Qtraces(i) \\ \text{implies} & (* \text{ lemma A.13.2 } *) \\ & \delta \in out(\, i \,\textbf{after}\, \sigma\,) \\ \text{implies} & (* \text{ premiss } *) \\ & \delta \in out(\, s \,\textbf{after}\, \sigma\,) \\ \text{implies} & (* \text{ lemma A.13.2 } *) \\ & \sigma \in Qtraces(s) \end{array}$$

□

A.3 Proofs of Section 4.2 (Implementation relations with inputs and outputs — Input-output refusal relation)

The proof of proposition 4.11 is analogous to the proof of proposition 3.8. We start with properties of the transformation Θ (definition A.6) and of the transition relations $\stackrel{\varphi}{\Longrightarrow}$ and $\stackrel{\varphi}{\Longrightarrow\!\!\!\!\succ}$ when applied to input-output transition systems. Then the analogue of lemma A.8 for input-output transition systems is given in lemma A.16. Definition A.17 defines a special observer which can observe a particular suspension trace. Suspension traces and observations are related in lemma A.18. Proposition 4.11 is then a straightforward consequence.

Lemma A.14
Let $i \in \mathcal{IOTS}(L_I, L_U)$, $u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$, and $\varphi \in (L \cup \mathcal{P}(L))^*$.

1. If $i \stackrel{\varphi}{\Longrightarrow}$ then $\varphi \in (L \cup \mathcal{P}(L_U))^*$
2. If $u \stackrel{\varphi}{\Longrightarrow\!\!\!\!\succ}$ then $\varphi \in (L \cup \{A \mid L_U \subseteq A \subseteq L\})^*$

$\square$

Proof (lemma A.14)
Both proofs are by induction on the structure of φ, where the cases $\varphi = \epsilon$ and $\varphi = a{\cdot}\varphi'$ $(a \in L)$ are trivial. The remaining cases are those with $\varphi = A{\cdot}\varphi'$ $(A \subseteq L)$.

1. Let $i \stackrel{A\cdot\varphi'}{\Longrightarrow}$ then $\exists i_1, i_2 :\ i \stackrel{\epsilon}{\Longrightarrow} i_1 \stackrel{A}{\longrightarrow} i_2 \stackrel{\varphi'}{\Longrightarrow}$. From definition 3.7: $\forall \mu \in A \cup \{\tau\} : i_1 \stackrel{\mu}{\not\longrightarrow}$, and since $i \in \mathcal{IOTS}(L_I, L_U)$: $\forall a \in L_I : i_1 \stackrel{a}{\Longrightarrow}$, so $A \subseteq L_U$. Together with induction, $\varphi' \in (L \cup \mathcal{P}(L_U))^*$, we have $A{\cdot}\varphi' \in (L \cup \mathcal{P}(L_U))^*$.
2. Let $u \stackrel{A\cdot\varphi'}{\Longrightarrow\!\!\!\!\succ}$ then $\exists u_1, u_2 :\ u \stackrel{\epsilon}{\Longrightarrow} u_1 \stackrel{A}{\longrightarrow\!\!\!\!\succ} u_2 \stackrel{\varphi'}{\Longrightarrow}$. From definition A.5: $\mathit{init}(u_1) = A \cup \{\theta\}$, and since $u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$: $\forall x \in L_U : u_1 \stackrel{x}{\Longrightarrow}$, so $L_U \subseteq A$. Together with induction, $\varphi' \in (L \cup \{A \mid L_U \subseteq A \subseteq L\})^*$, we have $A{\cdot}\varphi' \in (L \cup \{A \mid L_U \subseteq A \subseteq L\})^*$.

$\square$

Lemma A.15
The transformation Θ, when restricted to the domain $(L \cup \{L_U\})^*$, is a bijection.

$\square$

Proof (lemma A.15)
Define $\Theta^{-1} : L_\theta^* \to (L \cup \{L_U\})^*$ by

$$\begin{array}{lll} \Theta^{-1}(\epsilon) & =_{\mathrm{def}} & \epsilon \\ \Theta^{-1}(a{\cdot}\sigma) & =_{\mathrm{def}} & a{\cdot}\Theta^{-1}(\sigma) \\ \Theta^{-1}(\theta{\cdot}\sigma) & =_{\mathrm{def}} & L_U{\cdot}\Theta(\sigma) \end{array}$$

then for each $\sigma \in L_\theta$, $\Theta^{-1}(\sigma)$ is defined, and from definition A.6 it is clear that $\Theta^{-1}(\Theta(\varphi)) = \varphi$ for each $\varphi \in (L \cup \{L_U\})^*$.

$\square$

Lemma A.16
Let $u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$, $i \in \mathcal{IOTS}(L_I, L_U)$, $r \in \mathcal{LTS}(L_\theta)$, and $\sigma \in L_\theta^*$, then

$$u \rceil\!| \, i \stackrel{\sigma}{\Longrightarrow} r \quad \text{implies} \quad \exists u', i' :\ u \stackrel{\Theta^{-1}(\sigma)}{\Longrightarrow\!\!\!\!\succ} u' \ \text{ and } \ i \stackrel{\Theta^{-1}(\sigma)}{\Longrightarrow} i' \ \text{ and } \ r = u' \rceil\!| \, i'$$

$\square$

Proof (lemma A.16)
From lemma A.8.1:

$$\begin{array}{lll} u \rceil\!| \, i \stackrel{\sigma}{\Longrightarrow} r & \text{implies} & \exists u', i',\ \exists \varphi \in (L \cup \mathcal{P}(L))^* : \\ & & u \stackrel{\varphi}{\Longrightarrow\!\!\!\!\succ} u' \ \text{ and } \ i \stackrel{\varphi}{\Longrightarrow} i' \ \text{ and } \ r = u' \rceil\!| \, i' \ \text{ and } \ \Theta(\varphi) = \sigma \end{array}$$

Using lemmata A.14.1, and A.14.2: $\varphi \in (L \cup \{L_U\})^*$. Then, using lemma A.15: $\varphi = \Theta^{-1}(\sigma)$ $\square$

Definition A.17
Let $\varphi \in (L \cup \{L_U\})^*$ be a suspension trace, then $\overline{u}_{[\varphi]}$ is defined as follows, where $a \in L_I$, $x \in L_U$, and the process C_U is defined by $C_U := \Sigma\{x; C_U \mid x \in L_U\}$:

$$\begin{array}{lll}
\overline{u}_{[\epsilon]} & =_{\mathrm{def}} & C_U \\
\overline{u}_{[a\cdot\varphi]} & =_{\mathrm{def}} & C_U \,\Box\, a \;;\; \overline{u}_{[\varphi]} \\
\overline{u}_{[x\cdot\varphi]} & =_{\mathrm{def}} & \Sigma\{x; C_U \mid x \in L_U \backslash \{x\}\} \,\Box\, x \;;\; \overline{u}_{[\varphi]} \\
\overline{u}_{[L_U\cdot\varphi]} & =_{\mathrm{def}} & C_U \,\Box\, \theta; \overline{u}_{[\varphi]}
\end{array}$$

□

Lemma A.18
Let $i \in \mathcal{IOTS}(L_I, L_U)$, $\varphi \in (L \cup \{L_U\})^*$, and $\psi \in (L \cup \mathcal{P}(L))^*$.

1. $\overline{u}_{[\varphi]} \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$
2. $\overline{u}_{[\varphi]} \overset{\varphi}{\Longrightarrow}$
3. $\overline{u}_{[\varphi]} \overset{\psi}{\Longrightarrow}$ and $\Theta(\varphi) = \Theta(\psi)$ implies $\varphi = \psi$
4. $\varphi \in \mathit{Straces}(i)$ iff $\Theta(\varphi) \in \mathit{obs}_t^\theta(\overline{u}_{[\varphi]}, i)$

□

Proof (lemma A.18)

1. First, observe that $C_U \xrightarrow{x} C_U$ for all $x \in L_U$, so $C_U \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$. Then, by induction on the structure of φ, with $a \in L_I$ and $x \in L_U$:

   $\varphi = \epsilon$: $\overline{u}_{[\epsilon]} = C_U \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$

   $\varphi = a\cdot\varphi'$: $\overline{u}_{[a\cdot\varphi']} = C_U \,\Box\, a \;;\; \overline{u}_{[\varphi']} \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$ if $\overline{u}_{[\varphi']} \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$

   $\varphi = x\cdot\varphi'$: $\overline{u}_{[x\cdot\varphi']} = \Sigma\{x; C_U \mid x \in L_U\backslash\{x\}\} \,\Box\, x \;;\; \overline{u}_{[\varphi']} \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$
   if $\overline{u}_{[\varphi']} \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$

   $\varphi = L_U\cdot\varphi'$: $\overline{u}_{[L_U\cdot\varphi']} = C_U \,\Box\, \theta; \overline{u}_{[\varphi']} \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$ if $\overline{u}_{[\varphi']} \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$

2. By induction on the structure of φ, with $a \in L_I$ and $x \in L_U$:

   $\varphi = \epsilon$: $\overline{u}_{[\epsilon]} = C_U \overset{\epsilon}{\Longrightarrow}$

   $\varphi = a\cdot\varphi'$: $\overline{u}_{[a\cdot\varphi']} = C_U \,\Box\, a \;;\; \overline{u}_{[\varphi']} \xrightarrow{a} \overline{u}_{[\varphi']}$ and $\overline{u}_{[\varphi']} \overset{\varphi'}{\Longrightarrow}$ by induction, so $\overline{u}_{[a\cdot\varphi']} \overset{a\cdot\varphi'}{\Longrightarrow}$

   $\varphi = x\cdot\varphi'$: $\overline{u}_{[x\cdot\varphi']} = \Sigma\{x; C_U \mid x \in L_U\backslash\{x\}\} \,\Box\, x \;;\; \overline{u}_{[\varphi']} \xrightarrow{x} \overline{u}_{[\varphi']}$ and $\overline{u}_{[\varphi']} \overset{\varphi'}{\Longrightarrow}$ by induction, so $\overline{u}_{[x\cdot\varphi']} \overset{x\cdot\varphi'}{\Longrightarrow}$

   $\varphi = L_U\cdot\varphi'$: $\overline{u}_{[L_U\cdot\varphi']} = C_U \,\Box\, \theta; \overline{u}_{[\varphi']} \overset{L_U}{\Longrightarrow} \overline{u}_{[\varphi']}$ and $\overline{u}_{[\varphi']} \overset{\varphi'}{\Longrightarrow}$ by induction,
   so $\overline{u}_{[L_U\cdot\varphi']} \overset{L_U\cdot\varphi'}{\Longrightarrow}$

3. By induction on the structure of ψ, with $a \in L$, $A \subseteq L$:

   $\psi = \epsilon$: $\Theta(\psi) = \Theta(\epsilon) = \epsilon$, so $\Theta(\varphi) = \epsilon$, hence $\varphi = \epsilon$.

   $\psi = a\cdot\psi'$:

$$\begin{array}{ll}
 & \overline{u}_{[\varphi]} \overset{a\cdot\psi'}{\Longrightarrow} \text{ and } \Theta(\varphi) = \Theta(a\cdot\psi') \\
\text{implies} & (*\text{ definition A.17 }*) \\
 & \exists\varphi' : \varphi = a\cdot\varphi' \text{ and } \overline{u}_{[\varphi]} = \overline{u}_{[a\cdot\varphi']} \overset{a\cdot\psi'}{\Longrightarrow} \text{ and } \Theta(a\cdot\varphi') = \Theta(a\cdot\psi') \\
\text{implies} & (*\text{ definitions A.17, A.6 }*) \\
 & \exists\varphi' : \varphi = a\cdot\varphi' \text{ and } \overline{u}_{[\varphi']} \overset{\psi'}{\Longrightarrow} \text{ and } \Theta(\varphi') = \Theta(\psi') \\
\text{implies} & (*\text{ induction }*) \\
 & \exists\varphi' : \varphi = a\cdot\varphi' \text{ and } \varphi' = \psi' \\
\text{implies} & \varphi = a\cdot\psi' = \psi
\end{array}$$

$\psi = A{\cdot}\psi'$:

$$\begin{array}{ll}
 & \overline{u}_{[\varphi]} \overset{A\cdot\psi'}{\Longrightarrow\!\!\!\!\succ} \text{ and } \Theta(\varphi) = \Theta(A{\cdot}\psi') \\
\text{implies} & (* \text{ definitions A.17 and A.5 } *) \\
 & A = L_U \text{ and } \exists \varphi' : \varphi = L_U{\cdot}\varphi' \text{ and} \\
 & \overline{u}_{[\varphi]} = \overline{u}_{[L_U\cdot\varphi']} \overset{L_U\cdot\psi'}{\Longrightarrow\!\!\!\!\succ} \text{ and } \Theta(L_U{\cdot}\varphi') = \Theta(A{\cdot}\psi') \\
\text{implies} & (* \text{ definitions A.17 and A.6 } *) \\
 & \exists \varphi' : \varphi = L_U{\cdot}\varphi' \text{ and } u_{[\varphi']} \overset{\psi'}{\Longrightarrow\!\!\!\!\succ} \text{ and } \Theta(\varphi') = \Theta(\psi') \text{ and } A = L_U \\
\text{implies} & (* \text{ induction } *) \\
 & \exists \varphi' : \varphi = L_U{\cdot}\varphi' \text{ and } \varphi' = \psi' \text{ and } A = L_U \\
\text{implies} & \varphi = A{\cdot}\psi' = \psi
\end{array}$$

4.

$$\begin{array}{ll}
 & \varphi \in \mathit{Straces}(i) \\
\text{iff} & (* \text{ definition 4.10 and lemma A.18.2 } *) \\
 & i \overset{\varphi}{\Longrightarrow} \text{ and } \varphi \in (L \cup \{L_U\})^* \text{ and } \overline{u}_{[\varphi]} \overset{\varphi}{\Longrightarrow\!\!\!\!\succ} \\
\text{iff} & (* \text{ lemmata A.16 and A.8.2 } *) \\
 & \overline{u}_{[\varphi]} \rceil\!| \, i \overset{\Theta(\varphi)}{\Longrightarrow} \\
\text{iff} & (* \text{ definition 3.6.2 } *) \\
 & \Theta(\varphi) \in \mathit{obs}_t^\theta(\overline{u}_{[\varphi]}, i)
\end{array}$$

□

Lemma A.19
Let $i \in \mathcal{IOTS}(L_I, L_U)$, $s \in \mathcal{LTS}(L_I \cup L_U)$, then

$$\mathit{Straces}(i) \subseteq \mathit{Straces}(s) \quad \text{implies} \quad \forall u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\}) : \ \mathit{obs}_c^\theta(u, i) \subseteq \mathit{obs}_c^\theta(u, s)$$

□

Proof (lemma A.19)
Let $u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$, $\sigma \in L_\theta^*$, then

$$\begin{array}{ll}
 & \sigma \in \mathit{obs}_c^\theta(u, i) \\
\text{implies} & (* \text{ definitions 3.6.2, 3.2.3 and claim (2) } *) \\
 & \exists r : \ u \rceil\!| \, i \overset{\sigma}{\Longrightarrow} r \text{ and } \forall \mu \in L_\theta \cup \{\tau\} : r \overset{\mu}{\not\longrightarrow} \\
\text{implies} & (* \text{ lemma A.16 } *) \\
 & \exists u', i' : \ u \overset{\Theta^{-1}(\sigma)}{\Longrightarrow\!\!\!\!\succ} u' \text{ and } i \overset{\Theta^{-1}(\sigma)}{\Longrightarrow} i' \text{ and } \forall \mu \in L_\theta \cup \{\tau\} : u' \rceil\!| \, i' \overset{\mu}{\not\longrightarrow} \\
\text{implies} & (* \text{ definitions 2.4 and 3.6.1 } *) \\
 & \exists u', i' : \ u \overset{\Theta^{-1}(\sigma)}{\Longrightarrow\!\!\!\!\succ} u' \text{ and } i \overset{\Theta^{-1}(\sigma)}{\Longrightarrow} i' \text{ and } \mathit{init}(u') = L_U \text{ and } \mathit{init}(i') = L_I \\
\text{implies} & (* \text{ definition 3.7 } *) \\
 & \exists u' : \ u \overset{\Theta^{-1}(\sigma)}{\Longrightarrow\!\!\!\!\succ} u' \text{ and } \mathit{init}(u') = L_U \text{ and } i \overset{\Theta^{-1}(\sigma)\cdot L_U}{\Longrightarrow} \\
\text{implies} & (* \text{ definition 4.10, lemma A.15, premiss } *) \\
 & \exists u' : \ u \overset{\Theta^{-1}(\sigma)}{\Longrightarrow\!\!\!\!\succ} u' \text{ and } \mathit{init}(u') = L_U \text{ and } s \overset{\Theta^{-1}(\sigma)\cdot L_U}{\Longrightarrow} \\
\text{implies} & (* \text{ definition 3.7 } *) \\
 & \exists u', s' : \ u \overset{\Theta^{-1}(\sigma)}{\Longrightarrow\!\!\!\!\succ} u' \text{ and } \mathit{init}(u') = L_U \text{ and} \\
 & s \overset{\Theta^{-1}(\sigma)}{\Longrightarrow} s' \text{ and } \forall \mu \in L_U \cup \{\tau\} : s' \overset{\mu}{\not\longrightarrow} \\
\text{implies} & (* \text{ lemmata A.8.2 and A.15, definition 3.6.1 } *) \\
 & \exists u', s' : \ u \rceil\!| \, s \overset{\sigma}{\Longrightarrow} u' \rceil\!| \, s' \text{ and } \forall \mu \in L_\theta \cup \{\tau\} : u' \rceil\!| \, s' \overset{\mu}{\not\longrightarrow} \\
\text{implies} & (* \text{ definitions 3.2.2, 3.6.2, and claim (2) } *) \\
 & \sigma \in \mathit{obs}_c^\theta(u, s)
\end{array}$$

□

Lemma A.20
Let $i \in \mathcal{IOTS}(L_I, L_U)$, $s \in \mathcal{LTS}(L_I \cup L_U)$, then

$$\mathit{Straces}(i) \subseteq \mathit{Straces}(s) \quad \text{iff} \quad \forall u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\}) : \ \mathit{obs}_t^\theta(u, i) \subseteq \mathit{obs}_t^\theta(u, s)$$

□

Proof (lemma A.20)

only if: Let $u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\})$, $\sigma \in L_\theta^*$, then

$$\begin{array}{ll}
 & \sigma \in obs_t^\theta(u,i) \\
\text{implies} & (*\ \text{definition 3.6.2}\ *) \\
 & u\,]|\, i \overset{\sigma}{\Longrightarrow} \\
\text{implies} & (*\ \text{lemma A.16}\ *) \\
 & u \overset{\Theta^{-1}(\sigma)}{\Longrightarrow\!\!\!\succ} \quad \text{and} \quad i \overset{\Theta^{-1}(\sigma)}{\Longrightarrow} \\
\text{implies} & (*\ \text{definition 4.10, lemma A.15, premiss}\ *) \\
 & u \overset{\Theta^{-1}(\sigma)}{\Longrightarrow\!\!\!\succ} \quad \text{and} \quad s \overset{\Theta^{-1}(\sigma)}{\Longrightarrow} \\
\text{implies} & (*\ \text{lemmata A.8.2 and A.15}\ *) \\
 & u\,]|\, s \overset{\sigma}{\Longrightarrow} \\
\text{implies} & (*\ \text{definition 3.6.2}\ *) \\
 & \sigma \in obs_t^\theta(u,s)
\end{array}$$

if:

$$\begin{array}{ll}
 & \varphi \in \mathit{Straces}(i) \\
\text{implies} & (*\ \text{lemma A.18.4 and definition 4.10}\ *) \\
 & \Theta(\varphi) \in obs_t^\theta(\overline{u}_{[\varphi]}, i) \ \ \text{and} \ \ \varphi \in (L \cup \{L_U\})^* \\
\text{implies} & (*\ \text{premiss and lemma A.18.1}\ *) \\
 & \Theta(\varphi) \in obs_t^\theta(\overline{u}_{[\varphi]}, s) \ \ \text{and} \ \ \varphi \in (L \cup \{L_U\})^* \\
\text{implies} & (*\ \text{definition 3.6.2}\ *) \\
 & \overline{u}_{[\varphi]}\,]|\, s \overset{\Theta(\varphi)}{\Longrightarrow} \quad \text{and} \ \ \varphi \in (L \cup \{L_U\})^* \\
\text{implies} & (*\ \text{lemma A.8.1}\ *) \\
 & \exists \psi \in (L \cup \mathcal{P}(L))^* : \ \overline{u}_{[\varphi]} \overset{\psi}{\Longrightarrow\!\!\!\succ} \quad \text{and} \quad s \overset{\psi}{\Longrightarrow} \quad \text{and} \\
 & \Theta(\psi) = \Theta(\varphi) \ \ \text{and} \ \ \varphi \in (L \cup \{L_U\})^* \\
\text{implies} & (*\ \text{lemma A.18.3}\ *) \\
 & s \overset{\varphi}{\Longrightarrow} \quad \text{and} \ \ \varphi \in (L \cup \{L_U\})^* \\
\text{implies} & (*\ \text{definition 4.10}\ *) \\
 & \varphi \in \mathit{Straces}(s)
\end{array}$$

□

Proposition 4.11

$i \leq_{ior} s$ iff $\mathit{Straces}(i) \subseteq \mathit{Straces}(s)$

□

Proof (proposition 4.11)

$$\begin{array}{ll}
 & i \leq_{ior} s \\
\text{iff} & (*\ \text{definition 4.9}\ *) \\
 & \forall u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\}) : \quad obs_c^\theta(u,i) \subseteq obs_c^\theta(u,s) \ \ \text{and} \ \ obs_t^\theta(u,i) \subseteq obs_t^\theta(u,s) \\
\text{iff} & (*\ \text{lemma A.20}\ *) \\
 & \forall u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\}) : \quad obs_c^\theta(u,i) \subseteq obs_c^\theta(u,s) \ \ \text{and} \ \ \mathit{Straces}(i) \subseteq \mathit{Straces}(s) \\
\text{iff} & (*\ \text{lemma A.19}\ *) \\
 & \mathit{Straces}(i) \subseteq \mathit{Straces}(s)
\end{array}$$

□

Lemma A.21

Let $p \in \mathcal{LTS}(L)$, $\sigma \in L_\delta^*$, $x \in L_U \cup \{\delta\}$, then

1. $x \in out(\, p \textbf{ after } \sigma\,)$ iff $\sigma \cdot x \in \mathit{Straces}(p)$
2. $out(\, p \textbf{ after } \sigma\,) \neq \emptyset$ iff $\sigma \in \mathit{Straces}(p)$

□

Proof (lemma A.21)

1. $x \in out(\, p\,\mathbf{after}\,\sigma\,)$

iff (∗ definition 4.4.2 ∗)

$\exists p' \in (\, p\,\mathbf{after}\,\sigma\,):\; x \in out(p')$

iff (∗ definition 4.4.1 ∗)

$\exists p' \in (\, p\,\mathbf{after}\,\sigma\,):\; (\; x \in L_U \;\text{ and }\; p' \xrightarrow{x} \;)\;\text{ or }\;(\; x = \delta \;\text{ and }\; \delta(p')\;)$

iff (∗ definitions 2.3 and 4.2.1 ∗)

$(\; x \in L_U \;\text{ and }\; \exists p' : p \stackrel{\sigma}{\Longrightarrow} p' \xrightarrow{x} \;)\;\text{ or}$

$(\; x = \delta \;\text{ and }\; \exists p' : p \stackrel{\sigma}{\Longrightarrow} p' \text{ and } \forall \mu \in L_U \cup \{\tau\} : p' \stackrel{\mu}{\not\longrightarrow} \;)$

iff (∗ definitions 2.2 and 3.7 ∗)

$(\; x \in L_U \;\text{ and }\; p \stackrel{\sigma\cdot x}{\Longrightarrow} \;)\;\text{ or }\;(\; x = \delta \;\text{ and }\; p \stackrel{\sigma\cdot L_U}{\Longrightarrow} \;)$

iff (∗ definition 4.10 ∗)

$\sigma\cdot x \in \mathit{Straces}(p)$

2. *only if*:

$out(\, p\,\mathbf{after}\,\sigma\,) \neq \emptyset$

implies $\exists x \in L_U \cup \{\delta\}:\; x \in out(\, p\,\mathbf{after}\,\sigma\,)$

implies (∗ lemma A.21.1 ∗)

$\exists x \in L_U \cup \{\delta\}:\; \sigma\cdot x \in \mathit{Straces}(p)$

implies (∗ definitions 2.2 and 2.3 ∗)

$\sigma \in \mathit{Straces}(p)$

if:

$\sigma \in \mathit{Straces}(p)$

implies (∗ definition 2.3 and standard logic ∗)

$\exists p' : p \stackrel{\sigma}{\Longrightarrow} p' \;\text{ and }\;(\; \forall x \in L_U : p' \stackrel{x}{\not\Longrightarrow} \;\text{ or }\; \exists x \in L_U : p' \stackrel{x}{\Longrightarrow} \;)$

implies (∗ definition 3.2.1 and claim (2), definition 2.2 ∗)

$(\; \exists p' : p \stackrel{\sigma}{\Longrightarrow} p' \text{ and } \forall \mu \in L_U \cup \{\tau\} : p' \stackrel{\mu}{\not\longrightarrow} \;)\;\text{ or}$

$(\; \exists p' : p \stackrel{\sigma}{\Longrightarrow} p' \text{ and } \exists x \in L_U, \exists p'' : p' \stackrel{\epsilon}{\Longrightarrow} p'' \xrightarrow{x} \;)$

implies (∗ definitions 2.3, 4.2.1 and 2.2 ∗)

$(\; \exists p' \in (\, p\,\mathbf{after}\,\sigma\,):\; \delta(p')\;)\;\text{ or}$

$(\; \exists p'' \in (\, p\,\mathbf{after}\,\sigma\,):\; \exists x \in L_U :\; p'' \xrightarrow{x} \;)$

implies (∗ definition 4.4.1 ∗)

$(\; \exists p' \in (\, p\,\mathbf{after}\,\sigma\,):\; \delta \in out(p')\;)\;\text{ or}$

$(\; \exists p'' \in (\, p\,\mathbf{after}\,\sigma\,):\; \exists x \in L_U :\; x \in out(p''))$

implies (∗ definition 4.4.2 ∗)

$out(\, p\,\mathbf{after}\,\sigma\,) \neq \emptyset$

□

Proposition 4.12

$i \leq_{ior} s \quad\text{iff}\quad \forall \sigma \in L_\delta^* :\; out(\, i\,\mathbf{after}\,\sigma\,) \;\subseteq\; out(\, s\,\mathbf{after}\,\sigma\,)$

□

Proof (proposition 4.12)

only if: Let $\sigma \in L_\delta^*$, then

$x \in out(\, i\,\mathbf{after}\,\sigma\,)$

implies (∗ lemma A.21.1 ∗)

$\sigma\cdot x \in \mathit{Straces}(i)$

implies (∗ premiss, proposition 4.11 ∗)

$\sigma\cdot x \in \mathit{Straces}(s)$

implies (∗ lemma A.21.1 ∗)

$x \in out(\, s\,\mathbf{after}\,\sigma\,)$

if: Using proposition 4.11, then

$$\begin{array}{ll}
& \sigma \in Straces(i) \\
\text{implies} & (*\ \text{lemma A.21.2}\ *) \\
& out(\, i \, \mathbf{after} \, \sigma \,) \neq \emptyset \\
\text{implies} & (*\ \text{premiss}\ *) \\
& out(\, s \, \mathbf{after} \, \sigma \,) \neq \emptyset \\
\text{implies} & (*\ \text{lemma A.21.2}\ *) \\
& \sigma \in Straces(s)
\end{array}$$

$\square$

A.4 Proofs of Section 4.3 (Implementation relations with inputs and outputs — Relating relations with inputs and outputs)

Claim table 1
The claim expressed in table 1 is:

1. $i \leq_{te} s \;=_{\text{def}}\; \forall u \in \mathcal{LTS}(L):\; obs_c(u,i) \subseteq obs_c(u,s) \;\text{ and }\; obs_t(u,i) \subseteq obs_t(u,s)$
   $\qquad\qquad \text{iff} \;\; \forall u \in \mathcal{LTS}(L):\; obs_c(u,i) \subseteq obs_c(u,s)$

2. $i \leq_{rf} s \;=_{\text{def}}\; \forall u \in \mathcal{LTS}(L_\theta):\; obs_c^\theta(u,i) \subseteq obs_c^\theta(u,s) \;\text{ and }\; obs_t^\theta(u,i) \subseteq obs_t^\theta(u,s)$
   $\qquad\qquad \text{iff} \;\; \forall u \in \mathcal{LTS}(L_\theta):\; obs_c^\theta(u,i) \subseteq obs_c^\theta(u,s)$
   $\qquad\qquad \text{iff} \;\; \forall u \in \mathcal{LTS}(L_\theta):\; obs_t^\theta(u,i) \subseteq obs_t^\theta(u,s)$

3. $i \leq_{ior} s \;=_{\text{def}}\; \forall u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\}) : obs_c^\theta(u,i) \subseteq obs_c^\theta(u,s) \text{ and } obs_t^\theta(u,i) \subseteq obs_t^\theta(u,s)$
   $\qquad\qquad \text{iff} \;\; \forall u \in \mathcal{IOTS}(L_U, L_I \cup \{\theta\}) : obs_t^\theta(u,i) \subseteq obs_t^\theta(u,s)$

□

Proof (table 1)

1. Definition 3.2.4 and lemma A.3.
2. Definition 3.6.3, proposition 3.8 and lemmata A.11 and A.12.
3. Definition 4.9, proposition 4.11 and lemma A.20.

□

Claim (4)

1. $\leq_{iot} \;=\; \mathbf{ioco}_{L^*}$
2. $\leq_{ior} \;=\; \mathbf{ioco}_{L_\delta^*}$
3. $\mathbf{ioconf} \;=\; \mathbf{ioco}_{traces(s)}$
4. $\mathbf{ioco} \;=\; \mathbf{ioco}_{Straces(s)}$

□

Proof (claim (4))

1. Proposition 4.5.
2. Proposition 4.12.
3. Definition 4.7.
4. Definition 4.13.

□

Proposition 4.15

$$\leq_{ior} \;\subset\; \left\{ \begin{array}{c} \leq_{iot} \\ \mathbf{ioco} \end{array} \right\} \;\subset\; \mathbf{ioconf}$$

□

Proof (proposition 4.15)
The inclusions follow directly from (3) and (4) using

$$L_\delta^* \;\supseteq\; \left\{ \begin{array}{l} L^* \\ Straces(s) \end{array} \right\} \;\supseteq\; traces(s)$$

The inequalities follow from examples 4.8 and 4.14.

□

A.5 Proofs of Section 4.4 (Implementation relations with inputs and outputs — Suspension automata)

Lemma A.22
Let $p \in \mathcal{LTS}(L)$, $\sigma \in L_\delta^*$ and $a \in L$, then

1. $p\,\textbf{after}\,\sigma{\cdot}a \;=\; \{\; p'' \mid \exists p' \in (\,p\,\textbf{after}\,\sigma\,) :\; p' \stackrel{a}{\Longrightarrow} p'' \;\}$
2. $p\,\textbf{after}\,\sigma{\cdot}\delta \;=\; \{\; p' \mid p' \in (\,p\,\textbf{after}\,\sigma\,) \;\text{ and }\; \delta(p') \;\}$
3. $\Gamma_p \stackrel{\sigma}{\Longrightarrow} \Gamma'$ iff $\Gamma_p \stackrel{\sigma}{\longrightarrow} \Gamma'$
4. $\Gamma_p \stackrel{\sigma}{\Longrightarrow} \Gamma'$ iff $\Gamma' = p\,\textbf{after}\,\sigma$

□

Proof (lemma A.22)

1. $$\begin{array}{rl} & p\,\textbf{after}\,\sigma{\cdot}a \\ = & (*\;\text{ definition 2.3 }\;*) \\ & \{p'' \mid p \stackrel{\sigma\cdot a}{\Longrightarrow} p''\} \\ = & (*\;\text{ definition 2.2 }\;*) \\ & \{p'' \mid \exists p' : p \stackrel{\sigma}{\Longrightarrow} p' \stackrel{a}{\Longrightarrow} p''\} \\ = & (*\;\text{ definition 2.3 }\;*) \\ & \{p'' \mid \exists p' \in (\,p\,\textbf{after}\,\sigma\,) : p' \stackrel{a}{\Longrightarrow} p''\} \end{array}$$
2. $$\begin{array}{rl} & p\,\textbf{after}\,\sigma{\cdot}\delta \\ = & (*\;\text{ definition 2.3 }\;*) \\ & \{p' \mid p \stackrel{\sigma\cdot\delta}{\Longrightarrow} p'\} \\ = & (*\;\text{ definition 2.2 and 4.10 }\;*) \\ & \{p' \mid \exists p_1, p_2 : p \stackrel{\sigma}{\Longrightarrow} p_1 \stackrel{L_U}{\longrightarrow} p_2 \stackrel{\epsilon}{\Longrightarrow} p'\} \\ = & (*\;\text{ definition 3.7: } p_1 = p_2 = p'\;\;*) \\ & \{p' \mid p \stackrel{\sigma}{\Longrightarrow} p' \;\text{ and }\; \forall \mu \in L_U \cup \{\tau\} : p' \stackrel{\mu}{\not\longrightarrow}\} \\ = & (*\;\text{ definitions 2.3 and 4.2 }\;*) \\ & \{p' \mid p' \in (\,p\,\textbf{after}\,\sigma\,) \;\text{ and }\; \delta(p')\} \end{array}$$
3. Directly from definition 4.16: Γ_p does not contain any τ-transition.
4. By induction on the length of σ, with $\sigma' \in L_\delta^*$, $a \in L$:

   $\sigma = \epsilon$:
   $$\begin{array}{rl} & \Gamma_p \stackrel{\epsilon}{\Longrightarrow} \Gamma' \\ \text{iff} & (*\;\text{ lemma A.22.3 }\;*) \\ & \Gamma_p = \Gamma' \\ \text{iff} & (*\;\text{ definition 4.16 }\;*) \\ & \Gamma' = \{s' \mid p \stackrel{\epsilon}{\Longrightarrow} s'\} \\ \text{iff} & (*\;\text{ definition 2.3 }\;*) \\ & \Gamma' = p\,\textbf{after}\,\epsilon \end{array}$$

   $\sigma = \sigma'{\cdot}a$:
   $$\begin{array}{rl} & \Gamma_p \stackrel{\sigma'\cdot a}{\Longrightarrow} \Gamma' \\ \text{iff} & (*\;\text{ lemma A.22.3 }\;*) \\ & \Gamma_p \stackrel{\sigma'\cdot a}{\longrightarrow} \Gamma' \\ \text{iff} & (*\;\text{ definition 2.2 }\;*) \\ & \exists \Gamma_1 :\; \Gamma_p \stackrel{\sigma'}{\longrightarrow} \Gamma_1 \stackrel{a}{\longrightarrow} \Gamma' \\ \text{iff} & (*\;\text{ lemma A.22.3 }\;*) \\ & \exists \Gamma_1 :\; \Gamma_p \stackrel{\sigma'}{\Longrightarrow} \Gamma_1 \;\text{ and }\; \Gamma_1 \stackrel{a}{\longrightarrow} \Gamma' \end{array}$$

$$
\begin{array}{ll}
\text{iff} & (* \text{ induction and definition 4.16 } *) \\
& \exists \Gamma_1 : \Gamma_1 = p \,\textbf{after}\, \sigma' \quad \text{and} \quad \Gamma' = \{ s' \mid \exists s \in \Gamma_1 : s \overset{a}{\Longrightarrow} s' \} \\
\text{iff} & (* \text{ lemma A.22.1 } *) \\
& \Gamma' = p \,\textbf{after}\, \sigma' \cdot a
\end{array}
$$

$\sigma = \sigma' \cdot \delta$:

$$
\begin{array}{ll}
& \Gamma_p \overset{\sigma' \cdot \delta}{\Longrightarrow} \Gamma' \\
\text{iff} & (* \text{ lemma A.22.3 } *) \\
& \Gamma_p \xrightarrow{\sigma' \cdot \delta} \Gamma' \\
\text{iff} & (* \text{ definition 2.2 } *) \\
& \exists \Gamma_1 : \Gamma_p \xrightarrow{\sigma'} \Gamma_1 \xrightarrow{\delta} \Gamma' \\
\text{iff} & (* \text{ lemma A.22.3 } *) \\
& \exists \Gamma_1 : \Gamma_p \overset{\sigma'}{\Longrightarrow} \Gamma_1 \quad \text{and} \quad \Gamma_1 \xrightarrow{\delta} \Gamma' \\
\text{iff} & (* \text{ induction and definition 4.16 } *) \\
& \exists \Gamma_1 : \Gamma_1 = p \,\textbf{after}\, \sigma' \quad \text{and} \quad \Gamma' = \{ s \mid s \in \Gamma_1 \text{ and } \delta(s) \} \\
\text{iff} & (* \text{ lemma A.22.2 } *) \\
& \Gamma' = p \,\textbf{after}\, \sigma' \cdot \delta
\end{array}
$$

□

Proposition 4.17
Let $p \in \mathcal{LTS}(L)$ with inputs in L_I and outputs in L_U, let $\sigma \in L_\delta^*$, and consider δ as an output action of Γ_p; i.e., Γ_p has inputs in L_I and outputs in $L_U \cup \{\delta\}$; then

1. Γ_p is deterministic.
2. $\mathit{traces}(\Gamma_p) \;=\; \mathit{Straces}(p)$
3. $\mathit{out}(\,\Gamma_p \,\textbf{after}\, \sigma\,) \;=\; \mathit{out}(\,p \,\textbf{after}\, \sigma\,)$
4. $\sigma \in \mathit{traces}(\Gamma_p) \quad \text{iff} \quad \mathit{out}(\,\Gamma_p \,\textbf{after}\, \sigma\,) \neq \emptyset$

□

Proof (proposition 4.17)

1. If $\sigma \notin \mathit{traces}(\Gamma_p)$, then $|\,\Gamma_p \,\textbf{after}\, \sigma\,| = 0$. If $\sigma \in \mathit{traces}(\Gamma_p)$, then

$$
\begin{array}{ll}
& |\,\Gamma_p \,\textbf{after}\, \sigma\,| \\
= & (* \text{ definition 2.3 } *) \\
& |\{ \Gamma' \mid \Gamma_p \overset{\sigma}{\Longrightarrow} \Gamma' \}| \\
= & (* \text{ lemma A.22.4 } *) \\
& |\{\, p \,\textbf{after}\, \sigma \,\}| \\
= & 1
\end{array}
$$

2. Let $\sigma \in L_\delta^*$, then

$$
\begin{array}{ll}
& \sigma \in \mathit{traces}(\Gamma_p) \\
\text{iff} & (* \text{ definition 2.3 } *) \\
& \exists \Gamma' : \Gamma_p \overset{\sigma}{\Longrightarrow} \Gamma' \\
\text{iff} & (* \text{ lemma A.22.4 and definition 4.16 } *) \\
& \exists \Gamma' : \Gamma' = (\,p \,\textbf{after}\, \sigma\,) \quad \text{and} \quad \Gamma' \neq \emptyset \\
\text{iff} & (* \text{ standard set theory } *) \\
& \exists p' \in (\,p \,\textbf{after}\, \sigma\,) \\
\text{iff} & (* \text{ definition 2.3 } *) \\
& \exists p' : p \overset{\sigma}{\Longrightarrow} p' \\
\text{iff} & (* \text{ definition 2.3 } *) \\
& \sigma \in \mathit{Straces}(p)
\end{array}
$$

3. $\quad out(\,\Gamma_p\,\mathbf{after}\,\sigma\,)$

$= \quad$ (∗ definition 4.4.2 ∗)

$\quad \bigcup\,\{out(\Gamma') \mid \Gamma' \in (\,\Gamma_p\,\mathbf{after}\,\sigma\,)\}$

$= \quad$ (∗ lemma A.22.4 ∗)

$\quad \bigcup\,\{out(\Gamma') \mid \Gamma' = (\,p\,\mathbf{after}\,\sigma\,)\}$

$= \quad out(\,p\,\mathbf{after}\,\sigma\,)$

4. Let $\sigma \in L_\delta^*$, then

$\quad \sigma \in traces(\Gamma_p)$

iff $\quad$ (∗ proposition 4.17.2 ∗)

$\quad \sigma \in Straces(p)$

iff $\quad$ (∗ lemma A.21.2 ∗)

$\quad out(\,p\,\mathbf{after}\,\sigma\,) \neq \emptyset$

iff $\quad$ (∗ lemma 4.17.3 ∗)

$\quad out(\,\Gamma_p\,\mathbf{after}\,\sigma\,) \neq \emptyset$

□

Corollary 4.18

$i \leq_{ior} s \quad \text{iff} \quad \Gamma_i \leq_{tr} \Gamma_s$

□

Proof (corollary 4.18)

$\quad i \leq_{ior} s$

iff $\quad$ (∗ proposition 4.11 ∗)

$\quad Straces(i) \ \subseteq \ Straces(s)$

iff $\quad$ (∗ proposition 4.17.2 ∗)

$\quad traces(\Gamma_i) \ \subseteq \ traces(\Gamma_s)$

iff $\quad$ (∗ definition 3.1 ∗)

$\quad \Gamma_i \leq_{tr} \Gamma_s$

□

A.6 Proofs of Section 6 (Test generation for input-output transition systems)

Note A.23
If $\sigma \in L_\delta^*$ then $\overline{\sigma}$ corresponds formally with $\Theta(\sigma)$; if $\sigma \in L_\theta^*$ then $\overline{\sigma}$ corresponds with $\Theta^{-1}(\sigma)$. $\square$

Lemma A.24
Let $t \in \mathcal{TEST}(L_U, L_I)$, $\varphi \in (L \cup \mathcal{P}(L))^*$, then $t \overset{\varphi}{\Longrightarrow\!\!\!\succ} t'$ implies $t \xrightarrow{\Theta(\varphi)} t'$ and $\varphi \in (L \cup \{L_U\})^*$

$\square$

Proof (lemma A.24)
By induction on the length of φ, with $a \in L$ and $A \subseteq L$:

$\varphi = \epsilon$:
If $\varphi = \epsilon$ then $\Theta(\varphi) = \epsilon$ and $t = t'$ (t is deterministic; definition 5.1.1); hence trivially $t \xrightarrow{\epsilon} t'$ and $\epsilon \in (L \cup \{L_U\})^*$.

$\varphi = a{\cdot}\varphi'$:

$$
\begin{array}{ll}
 & t \overset{a\cdot\varphi'}{\Longrightarrow\!\!\!\succ} t' \\
\text{implies} & (*\ \text{definitions 5.1.1 (determinism) and 2.2}\ *) \\
 & \exists t_1 :\ t \overset{a}{\longrightarrow\!\!\!\succ} t_1 \ \text{ and }\ t_1 \overset{\varphi'}{\Longrightarrow\!\!\!\succ} t' \\
\text{implies} & (*\ \text{definition A.5 and induction}\ *) \\
 & \exists t_1 :\ t \xrightarrow{a} t_1 \ \text{ and }\ t_1 \xrightarrow{\Theta(\varphi')} t' \ \text{ and }\ \varphi' \in (L \cup \{L_U\})^* \\
\text{implies} & (*\ \text{definition A.6}\ *) \\
 & t \xrightarrow{\Theta(a\cdot\varphi')} t' \ \text{ and }\ a{\cdot}\varphi' \in (L \cup \{L_U\})^*
\end{array}
$$

$\varphi = A{\cdot}\varphi'$:

$$
\begin{array}{ll}
 & t \overset{A\cdot\varphi'}{\Longrightarrow\!\!\!\succ} t' \\
\text{implies} & (*\ \text{definitions 5.1.1 (determinism) and 2.2}\ *) \\
 & \exists t_1 :\ t \overset{A}{\longrightarrow\!\!\!\succ} t_1 \ \text{ and }\ t_1 \overset{\varphi'}{\Longrightarrow\!\!\!\succ} t' \\
\text{implies} & (*\ \text{definition A.5 and induction}\ *) \\
 & \exists t_1 :\ \mathit{init}(t) = A \cup \{\theta\} \ \text{ and }\ t \xrightarrow{\theta} t_1 \ \text{ and }\ t_1 \xrightarrow{\Theta(\varphi')} t' \ \text{ and }\ \varphi' \in (L \cup \{L_U\})^* \\
\text{implies} & (*\ \text{definition 5.1.1}\ *) \\
 & A = L_U \ \text{ and }\ t \xrightarrow{\theta\cdot\Theta(\varphi')} t' \ \text{ and }\ \varphi' \in (L \cup \{L_U\})^* \\
\text{implies} & (*\ \text{definition A.6}\ *) \\
 & t \xrightarrow{\Theta(A\cdot\varphi')} t' \ \text{ and }\ A{\cdot}\varphi' \in (L \cup \{L_U\})^*
\end{array}
$$

$\square$

Claim (5)
A test case $t \in \mathcal{TEST}(L_U, L_I)$ is sound for a specification s with respect to $\mathbf{ioco}_\mathcal{F}$ if

$$
\begin{array}{ll}
\forall \sigma \in L_\theta^* : & t \xrightarrow{\sigma} \mathbf{fail} \ \text{ implies} \\
 & \exists \sigma' \in \mathcal{F},\ x \in L_U \cup \{\delta\} :\ \sigma = \Theta(\sigma'{\cdot}x) \ \text{ and }\ x \notin \mathit{out}(\,\Gamma_s \ \mathbf{after}\ \sigma'\,)
\end{array}
$$

$\square$

Proof (claim (5))
By contradiction: suppose that t is not sound and that the condition holds, then

$$
\begin{array}{ll}
 & t \text{ is not sound for } s \text{ with respect to } \mathbf{ioco}_\mathcal{F} \\
\text{implies} & (*\ \text{definition 6.1}\ *) \\
 & \exists i :\ i\ \mathbf{ioco}_\mathcal{F}\ s \ \text{ and }\ i\ \mathbf{fails}\ t \\
\text{implies} & (*\ \text{definition 5.2}\ *) \\
 & \exists i :\ i\ \mathbf{ioco}_\mathcal{F}\ s \ \text{ and }\ \exists \sigma \in L_\theta^*,\ \exists i' :\ t \rceil\!| \, i \overset{\sigma}{\Longrightarrow} \mathbf{fail} \rceil\!| \, i'
\end{array}
$$

implies (* lemma A.8.1 *)
$\exists i:\ i\ \mathbf{ioco}_{\mathcal{F}}\ s$ and $\exists \sigma \in L_\theta^*,\ \exists i',\ \exists \varphi \in (L \cup \mathcal{P}(L))^*$:
$t \overset{\varphi}{\Longrightarrow\!\!\succ} \mathbf{fail}$ and $i \overset{\varphi}{\Longrightarrow} i'$ and $\Theta(\varphi) = \sigma$

implies (* lemmata A.24 and A.15 *)
$\exists i:\ i\ \mathbf{ioco}_{\mathcal{F}}\ s$ and $\exists \sigma \in L_\theta^*:\ t \xrightarrow{\sigma} \mathbf{fail}$ and $i \overset{\Theta^{-1}(\sigma)}{\Longrightarrow}$

implies (* condition *)
$\exists i:\ i\ \mathbf{ioco}_{\mathcal{F}}\ s$ and $\exists \sigma' \in \mathcal{F},\ x \in L_U \cup \{\delta\}:\ x \notin out(\,\Gamma_s\ \mathbf{after}\ \sigma'\,)$ and $i \overset{\sigma' \cdot x}{\Longrightarrow}$

implies (* proposition 4.17.3 and lemma A.21.1 *)
$\exists i:\ i\ \mathbf{ioco}_{\mathcal{F}}\ s$ and $\exists \sigma' \in \mathcal{F},\ x \in L_U \cup \{\delta\}$:
$x \notin out(\,s\ \mathbf{after}\ \sigma'\,)$ and $x \in out(\,i\ \mathbf{after}\ \sigma'\,)$

implies (* equation (3) *)
$\exists \sigma' \in \mathcal{F},\ x \in L_U \cup \{\delta\}:\ x \notin out(\,s\ \mathbf{after}\ \sigma'\,)$ and $x \in out(\,s\ \mathbf{after}\ \sigma'\,)$

implies false

$\Box$

Lemma A.25

Let Γ be a suspension automaton, let $\mathcal{F} \subseteq traces(\Gamma)$, let $\sigma \in \mathcal{F}$, and let $t_{[\sigma,\mathcal{F},\Gamma]}$ be defined by:

$$
\begin{array}{lll}
t_{[\epsilon,\mathcal{F},\Gamma]} & =_{\text{def}} & \Sigma\ \{\ x\ ;\ \mathbf{fail} \mid x \in L_U \cup \{\theta\},\ \overline{x} \notin out(\Gamma)\ \} \\
 & & \Box\ \ \Sigma\ \{\ x\ ;\ \mathbf{pass} \mid x \in L_U \cup \{\theta\},\ \overline{x} \in out(\Gamma)\ \} \\[1em]
t_{[b\cdot\sigma,\mathcal{F},\Gamma]}\ \ (b \in L_I) & =_{\text{def}} & b\ ;\ t_{[\sigma,\mathcal{F}',\Gamma']} \\
 & & \quad \text{where}\ \ \mathcal{F}' = \{\sigma' \in L_\delta^* \mid b\cdot\sigma' \in \mathcal{F}\}\ \ \text{and}\ \ \Gamma \xrightarrow{b} \Gamma' \\[1em]
t_{[\overline{y}\cdot\sigma,\mathcal{F},\Gamma]}\ \ (y \in L_U \cup \{\theta\}) & =_{\text{def}} & \Sigma\ \{\ x\ ;\ \mathbf{fail} \mid x \in L_U \cup \{\theta\},\ \overline{x} \notin out(\Gamma),\ \epsilon \in \mathcal{F}\ \} \\
 & & \Box\ \ \Sigma\ \{\ x\ ;\ \mathbf{pass} \mid x \in L_U \cup \{\theta\},\ \overline{x} \notin out(\Gamma),\ \epsilon \notin \mathcal{F}\ \} \\
 & & \Box\ \ \Sigma\ \{\ x\ ;\ \mathbf{pass} \mid x \in L_U \cup \{\theta\},\ \overline{x} \in out(\Gamma),\ x \neq y\ \} \\
 & & \Box\ \ y\ ;\ t_{[\sigma,\mathcal{F}',\Gamma']} \\
 & & \quad \text{where}\ \ \mathcal{F}' = \{\sigma' \in L_\delta^* \mid \overline{y}\cdot\sigma' \in \mathcal{F}\}\ \ \text{and}\ \ \Gamma \xrightarrow{\overline{y}} \Gamma'
\end{array}
$$

then

1. $t_{[\sigma,\mathcal{F},\Gamma]}$ can be obtained from $\mathcal{F}$ and Γ with algorithm 6.2;
2. $x \notin out(\,\Gamma\ \mathbf{after}\ \sigma\,)$ implies $t_{[\sigma,\mathcal{F},\Gamma]} \overset{\sigma\cdot x}{\Longrightarrow\!\!\succ} \mathbf{fail}$

$\Box$

Proof (lemma A.25)

1. $t_{[\sigma,\mathcal{F},\Gamma]}$ can be obtained from $\mathcal{F}$ and Γ with algorithm 6.2 as follows:
   - $t_{[\epsilon,\mathcal{F},\Gamma]}$ is obtained with the third choice in algorithm 6.2, followed by the first choice for each t_x, using that $\epsilon = \sigma \in \mathcal{F}$.
   - $t_{[b\cdot\sigma,\mathcal{F},\Gamma]}$ $(b \in L_I)$ is obtained with the second choice in algorithm 6.2, choosing $a = b$, and followed by recursive application to obtain $t' = t_{[\sigma,\mathcal{F}',\Gamma']}$, using:
     - $\sigma \in \mathcal{F}' = \{\sigma' \in L_\delta^* \mid b\cdot\sigma' \in \mathcal{F}\} \neq \emptyset$ since $b\cdot\sigma \in \mathcal{F}$;
     - $\mathcal{F}' \subseteq traces(\Gamma')$ if $\mathcal{F} \subseteq traces(\Gamma)$.
   - $t_{[\overline{y}\cdot\sigma,\mathcal{F},\Gamma]}$ $(y \in L_U \cup \{\theta\})$ is obtained with the third choice in algorithm 6.2, followed by the first choice for each t_x with $x \neq y$, and recursive application to obtain $t_y = t_{[\sigma,\mathcal{F}',\Gamma']}$, using:
     - $\overline{y} \in out(\Gamma)$ since $\overline{y}\cdot\sigma \in \mathcal{F} \subseteq traces(\Gamma)$;
     - $\sigma \in \mathcal{F}' = \{\sigma' \in L_\delta^* \mid \overline{y}\cdot\sigma' \in \mathcal{F}\} \neq \emptyset$ since $\overline{y}\cdot\sigma \in \mathcal{F}$;
     - $\mathcal{F}' \subseteq traces(\Gamma')$ if $\mathcal{F} \subseteq traces(\Gamma)$.
2. By induction on σ:

$\sigma = \epsilon$:

$$
\begin{array}{ll}
 & x \notin out(\,\Gamma\ \textbf{after}\ \epsilon\,) \\
\text{implies} & (*\ \text{proposition 4.17.1}\ \ *) \\
 & x \notin out(\Gamma) \\
\text{implies} & (*\ \text{definition } t_{[\epsilon,\mathcal{F},\Gamma]}\ \ *) \\
 & t_{[\epsilon,\mathcal{F},\Gamma]} \xrightarrow{\overline{x}} \textbf{fail} \ \text{ and } \ init(t_{[\epsilon,\mathcal{F},\Gamma]}) = L_U \cup \{\theta\} \\
\text{implies} & (*\ \text{definition A.5}\ \ *) \\
 & t_{[\epsilon,\mathcal{F},\Gamma]} \overset{\epsilon\cdot x}{\Longrightarrow\!\!\!\succ} \textbf{fail}
\end{array}
$$

$\sigma = b\cdot\sigma'$, $b \in L_I$:

$$
\begin{array}{ll}
 & x \notin out(\,\Gamma\ \textbf{after}\ b\cdot\sigma'\,) \\
\text{implies} & (*\ \text{proposition 4.17.1, definition 2.3 and } b\cdot\sigma' \in \mathcal{F} \subseteq traces(\Gamma)\ \ *) \\
 & \exists\Gamma' :\ \Gamma \xrightarrow{b} \Gamma' \ \text{ and } \ x \notin out(\,\Gamma'\ \textbf{after}\ \sigma'\,) \\
\text{implies} & (*\ \text{definition } t_{[b\cdot\sigma',\mathcal{F},\Gamma]}\ \ *) \\
 & \exists\Gamma' :\ t_{[b\cdot\sigma',\mathcal{F},\Gamma]} \xrightarrow{b} t_{[\sigma',\mathcal{F}',\Gamma']} \ \text{ and } \ \mathcal{F}' = \{\sigma'' \in L_\delta^* \mid b\cdot\sigma'' \in \mathcal{F}\} \\
 & \ \text{and } \ x \notin out(\,\Gamma'\ \textbf{after}\ \sigma'\,) \\
\text{implies} & (*\ \text{induction, since} \\
 & \quad b\cdot\sigma' \in \mathcal{F} \subseteq traces(\Gamma) \text{ implies } \sigma' \in \mathcal{F}' \subseteq traces(\Gamma')\ \ *) \\
 & \exists\Gamma',\mathcal{F}' :\ t_{[b\cdot\sigma',\mathcal{F},\Gamma]} \xrightarrow{b} t_{[\sigma',\mathcal{F}',\Gamma']} \ \text{ and } \ t_{[\sigma',\mathcal{F}',\Gamma']} \overset{\sigma'\cdot x}{\Longrightarrow\!\!\!\succ} \textbf{fail} \\
\text{implies} & (*\ \text{definition A.5}\ \ *) \\
 & t_{[b\cdot\sigma',\mathcal{F},\Gamma]} \overset{b\cdot\sigma'\cdot x}{\Longrightarrow\!\!\!\succ} \textbf{fail}
\end{array}
$$

$\sigma = \overline{y}\cdot\sigma'$, $y \in L_U \cup \{\theta\}$:

$$
\begin{array}{ll}
 & x \notin out(\,\Gamma\ \textbf{after}\ \overline{y}\cdot\sigma'\,) \\
\text{implies} & (*\ \text{proposition 4.17.1, definition 2.3 and } \overline{y}\cdot\sigma \in \mathcal{F} \subseteq traces(\Gamma)\ \ *) \\
 & \exists\Gamma' :\ \Gamma \xrightarrow{\overline{y}} \Gamma' \ \text{ and } \ x \notin out(\,\Gamma'\ \textbf{after}\ \sigma'\,) \\
\text{implies} & (*\ \text{definition } t_{[\overline{y}\cdot\sigma',\mathcal{F},\Gamma]}\ \ *) \\
 & \exists\Gamma' :\ t_{[\overline{y}\cdot\sigma',\mathcal{F},\Gamma]} \xrightarrow{y} t_{[\sigma',\mathcal{F}',\Gamma']} \ \text{ and } \ \mathcal{F}' = \{\sigma'' \in L_\delta^* \mid \overline{y}\cdot\sigma'' \in \mathcal{F}\} \\
 & \ \text{and } \ x \notin out(\,\Gamma'\ \textbf{after}\ \sigma'\,) \\
\text{implies} & (*\ \text{induction, since} \\
 & \quad \overline{y}\cdot\sigma' \in \mathcal{F} \subseteq traces(\Gamma) \text{ implies } \sigma' \in \mathcal{F}' \subseteq traces(\Gamma')\ \ *) \\
 & \exists\Gamma',\mathcal{F}' :\ t_{[\overline{y}\cdot\sigma',\mathcal{F},\Gamma]} \xrightarrow{y} t_{[\sigma',\mathcal{F}',\Gamma']} \ \text{ and } \ t_{[\sigma',\mathcal{F}',\Gamma']} \overset{\sigma'\cdot x}{\Longrightarrow\!\!\!\succ} \textbf{fail} \\
\text{implies} & (*\ \text{definitions A.5 and } init(t_{[\overline{y}\cdot\sigma',\mathcal{F},\Gamma]} = L_U \cup \{\theta\}\ \ *) \\
 & t_{[\overline{y}\cdot\sigma',\mathcal{F},\Gamma]} \overset{\overline{y}\cdot\sigma'\cdot x}{\Longrightarrow\!\!\!\succ} \textbf{fail}
\end{array}
$$

□

Theorem 6.3
Let $s \in \mathcal{LTS}(L_I \cup L_U)$ and $\mathcal{F} \subseteq Straces(s)$; then

1. a test case obtained with algorithm 6.2 from Γ_s and $\mathcal{F}$ is sound for s with respect to $\textbf{ioco}_\mathcal{F}$;
2. the set of all possible test cases that can be obtained with algorithm 6.2 is exhaustive.

□

Proof (theorem 6.3)

1. By induction on the structure of t it is proved that each t generated with algorithm 6.2 from Γ and $\mathcal{F}$ satisfies the condition of claim (5):
   - Let $t = \textbf{pass}$, then the condition is trivially fulfilled.
   - Let $t = a;t'$ with $a \in L_I$, such that $\mathcal{F}' = \{\sigma \in L_\delta^* \mid a\cdot\sigma \in \mathcal{F}\} \neq \emptyset$ and t' is generated from $\mathcal{F}'$ and Γ', with $\Gamma \xrightarrow{a} \Gamma'$. Let $\sigma \in L_\theta^*$, such that $t \xrightarrow{\sigma} \textbf{fail}$, then it follows that $\sigma = a\cdot\sigma'$ ($\sigma' \in L_\theta^*$) and $t \xrightarrow{a} t' \xrightarrow{\sigma'} \textbf{fail}$. According to the induction hypothesis the condition can be assumed to hold for Γ', $\mathcal{F}'$, and t', hence

$$\exists\rho \in \mathcal{F}',\ \exists x \in L_U \cup \{\delta\} :\ \sigma' = \Theta(\rho\cdot x) \ \text{ and } \ x \notin out(\,\Gamma'\ \textbf{after}\ \rho\,)$$

It follows that $a{\cdot}\rho \in \mathcal{F}$ and
$x \in L_U \cup \{\delta\}$ and
$\sigma = a{\cdot}\sigma' = \Theta(a{\cdot}\rho{\cdot}x)$ and
$x \notin out(\,\Gamma\,\textbf{after}\,a{\cdot}\rho\,)$
which fulfil (5).

∘ Let $t \;=\; \Sigma\,\{\,x\,;\,\textbf{fail} \mid x \in L_U \cup \{\theta\},\ \overline{x} \notin out(\Gamma),\ \epsilon \in \mathcal{F}\}$
$\Box\ \Sigma\,\{\,x\,;\,\textbf{pass} \mid x \in L_U \cup \{\theta\},\ \overline{x} \notin out(\Gamma),\ \epsilon \notin \mathcal{F}\}$
$\Box\ \Sigma\,\{\,x\,;\,t_x \mid x \in L_U \cup \{\theta\},\ \overline{x} \in out(\Gamma)\ \}$;

where t_x is obtained recursively from $\mathcal{F}' = \{\sigma \in L_\delta^* \mid \overline{x}{\cdot}\sigma \in \mathcal{F}\}$ and Γ', with $\Gamma \xrightarrow{\overline{x}} \Gamma'$. Let $\sigma \in L_\theta^*$, such that $t \xrightarrow{\sigma} \textbf{fail}$, then it follows that $\sigma = y{\cdot}\sigma'$ ($\sigma' \in L_\theta^*$, $y \in L_U \cup \{\theta\}$) and $t \xrightarrow{y} t' \xrightarrow{\sigma'} \textbf{fail}$. Two cases are distinguished corresponding to the first summand and third summand of t:

- First summand:
Now $t \xrightarrow{y} \textbf{fail}$ and $\sigma' = \epsilon$, so $\overline{y} \notin out(\Gamma)$ and $\epsilon \in \mathcal{F}$;
hence it follows that $\epsilon \in \mathcal{F}$ and
$\overline{y} \in L_U \cup \{\delta\}$ and
$y = \Theta(\epsilon{\cdot}\overline{y})$ and
$\overline{y} \notin out(\,\Gamma\,\textbf{after}\,\epsilon\,)$
which fulfil (5).

- Third summand:
According to the induction hypothesis the condition can be assumed to hold for Γ', $\mathcal{F}'$, and t_y, hence

$$\exists \rho \in \mathcal{F}',\ \exists x \in L_U \cup \{\delta\}:\ \sigma' = \Theta(\rho{\cdot}x)\ \text{ and }\ x \notin out(\,\Gamma'\,\textbf{after}\,\rho\,)$$

It follows that $\overline{y}{\cdot}\rho \in \mathcal{F}$ and
$x \in L_U \cup \{\delta\}$ and
$\sigma = \Theta(\overline{y}{\cdot}\rho{\cdot}x)$ and
$x \notin out(\,\Gamma\,\textbf{after}\,\overline{y}{\cdot}\rho\,)$
which fulfil (5).

2. For exhaustiveness it must be proved that (definitions 6.1 and 5.2.3):

$$\forall i:\ i\ \cancel{\textbf{ioco}}_{\mathcal{F}}\ s\ \text{ implies }\ \exists t:\ t \text{ is obtained with algorithm 6.2 and } i\ \textbf{fails}\ t$$

Let $i \in \mathcal{IOTS}(L_I, L_U)$, then

$i\ \cancel{\textbf{ioco}}_{\mathcal{F}}\ s$
implies (∗ equation (3) ∗)
$\exists \sigma \in \mathcal{F}:\ out(\,i\,\textbf{after}\,\sigma\,) \not\subseteq out(\,s\,\textbf{after}\,\sigma\,)$
implies (∗ standard set theory ∗)
$\exists \sigma \in \mathcal{F},\ \exists x \in L_U \cup \{\delta\}:\ x \in out(\,i\,\textbf{after}\,\sigma\,)$ and $x \notin out(\,s\,\textbf{after}\,\sigma\,)$
implies (∗ lemma A.21.1, proposition 4.17.3 ∗)
$\exists \sigma \in \mathcal{F},\ \exists x \in L_U \cup \{\delta\}:\ i \xRightarrow{\sigma\cdot x}$ and $x \notin out(\,\Gamma_s\,\textbf{after}\,\sigma\,)$
implies (∗ lemma A.25.2 since $\sigma \in \mathcal{F} \subseteq Straces(s) = traces(\Gamma_s)$ ∗)
$\exists \sigma \in \mathcal{F},\ \exists x \in L_U \cup \{\delta\}:\ i \xRightarrow{\sigma\cdot x}$ and $t_{[\sigma,\mathcal{F},\Gamma_s]} \xRightarrow{\sigma\cdot x} \textbf{fail}$
implies (∗ lemma A.8.2 ∗)
$\exists \sigma \in \mathcal{F},\ \exists x \in L_U \cup \{\delta\},\ \exists i':\ t_{[\sigma,\mathcal{F},\Gamma_s]} \rceil\!| \, i \xRightarrow{\Theta(\sigma\cdot x)} \textbf{fail} \rceil\!| \, i'$
implies (∗ definition 5.2.2 ∗)
$\exists \sigma \in \mathcal{F}:\ i\ \textbf{fails}\ t_{[\sigma,\mathcal{F},\Gamma_s]}$
implies (∗ lemma A.25.1 ∗)
$\exists t:\ t$ is obtained with algorithm 6.2 and $i\ \textbf{fails}\ t$

□

Proposition 6.5
Let $\mathcal{F}$ be prefix-closed and let $\mathcal{G} = \mathcal{F} \cap (\mathit{Straces}(s) \cup \mathit{Straces}(s) \cdot L_I)$; then $\mathbf{ioco}_{\mathcal{F}} = \mathbf{ioco}_{\mathcal{G}}$ $\square$

Proof (proposition 6.5)

$\subseteq$: Directly from $\mathcal{G} \subseteq \mathcal{F}$.

$\supseteq$: By contraposition:

$$\begin{array}{ll} & \exists \sigma \in \mathcal{F} : \ \mathit{out}(\, i \,\mathbf{after}\, \sigma\,) \not\subseteq \mathit{out}(\, s \,\mathbf{after}\, \sigma\,) \\ \text{implies} & \exists \sigma \in \mathcal{G} : \ \mathit{out}(\, i \,\mathbf{after}\, \sigma\,) \not\subseteq \mathit{out}(\, s \,\mathbf{after}\, \sigma\,) \end{array}$$

$$\begin{array}{ll}
 & \exists \sigma \in \mathcal{F} : \ \mathit{out}(\, i \,\mathbf{after}\, \sigma\,) \not\subseteq \mathit{out}(\, s \,\mathbf{after}\, \sigma\,) \\
\text{implies} & (*\ \text{lemma A.21.1}\ *) \\
 & \exists \sigma \in \mathcal{F},\ \exists x \in L_U \cup \{\delta\} : \ i \overset{\sigma\cdot x}{\Longrightarrow} \ \text{and} \ s \overset{\sigma\cdot x}{\not\Longrightarrow} \\
\text{implies} & (*\ \text{definition 2.2: there is a maximum prefix } \sigma_1 \text{ of } \sigma\cdot x \text{ which } s \text{ can perform;} \\
 & \quad \sigma_1 \in \mathcal{F} \text{ since } \mathcal{F} \text{ is prefix-closed}\ *) \\
 & \exists \sigma \in \mathcal{F},\ \exists x \in L_U \cup \{\delta\} : \ i \overset{\sigma\cdot x}{\Longrightarrow} \ \text{and} \\
 & \exists \sigma_1, \sigma_2 \in L_\delta^*, a \in L : \ \sigma_1 \in \mathcal{F} \ \text{and} \ \sigma\cdot x = \sigma_1\cdot a\cdot\sigma_2 \ \text{and} \ s \overset{\sigma_1}{\Longrightarrow} \overset{a}{\not\Longrightarrow} \\
\text{implies} & (*\ \text{substitution for } \sigma; \\
 & \quad \text{distinguish between } a \in L_I \text{ and } a \in L_U \cup \{\delta\}; \\
 & \quad \text{if } a \in L_I \text{ then } \sigma_1 \neq \sigma \text{ so } \sigma_1\cdot a \text{ is a prefix of } \sigma, \text{ so } \sigma_1\cdot a \in \mathcal{F}\ *) \\
 & \quad (\ \exists \sigma_1 \in L_\delta^*, a \in L_I : \ \sigma_1\cdot a \in \mathcal{F} \ \text{and} \ i \overset{\sigma_1\cdot a}{\Longrightarrow} \ \text{and} \ s \overset{\sigma_1}{\Longrightarrow} \overset{a}{\not\Longrightarrow}\ ) \\
 & \text{or} \ (\ \exists \sigma_1 \in L_\delta^*, a \in L_U \cup \{\delta\} : \ \sigma_1 \in \mathcal{F} \ \text{and} \ i \overset{\sigma_1\cdot a}{\Longrightarrow} \ \text{and} \ s \overset{\sigma_1}{\Longrightarrow} \overset{a}{\not\Longrightarrow}\ ) \\
\text{implies} & (*\ \text{definition } \mathcal{G}\ *) \\
 & \quad (\ \exists \sigma_1 \in L_\delta^*, a \in L_I : \ \sigma_1\cdot a \in \mathcal{G} \ \text{and} \ i \overset{\sigma_1\cdot a}{\Longrightarrow} \ \text{and} \ s \overset{\sigma_1\cdot a}{\not\Longrightarrow}\ ) \\
 & \text{or} \ (\ \exists \sigma_1 \in L_\delta^*, a \in L_U \cup \{\delta\} : \ \sigma_1 \in \mathcal{G} \ \text{and} \ i \overset{\sigma_1\cdot a}{\Longrightarrow} \ \text{and} \ s \overset{\sigma_1\cdot a}{\not\Longrightarrow}\ ) \\
\text{implies} & (*\ \text{for } a \in L_I\text{: lemma A.21.2; for } a \in L_U \cup \{\delta\}\text{: lemma A.21.1}\ *) \\
 & \quad (\ \exists \sigma_1 \in L_\delta^*, a \in L_I : \ \sigma_1\cdot a \in \mathcal{G} \ \text{and} \\
 & \qquad \mathit{out}(\, i \,\mathbf{after}\, \sigma_1\cdot a\,) \neq \emptyset \ \text{and} \ \mathit{out}(\, s \,\mathbf{after}\, \sigma_1\cdot a\,) = \emptyset\ ) \\
 & \text{or} \ (\ \exists \sigma_1 \in L_\delta^*, a \in L_U \cup \{\delta\} : \ \sigma_1 \in \mathcal{G} \ \text{and} \\
 & \qquad a \in \mathit{out}(\, i \,\mathbf{after}\, \sigma_1\,) \ \text{and} \ a \notin \mathit{out}(\, s \,\mathbf{after}\, \sigma_1\,)\ ) \\
\text{implies} & (*\ \text{let } \sigma = \sigma_1\cdot a \text{ for } a \in L_I; \ \text{let } \sigma = \sigma_1 \text{ for } a \in L_U \cup \{\delta\}\ *) \\
 & \exists \sigma \in \mathcal{G} : \ \mathit{out}(\, i \,\mathbf{after}\, \sigma\,) \not\subseteq \mathit{out}(\, s \,\mathbf{after}\, \sigma\,)
\end{array}$$

$\square$