

Component Based Testing with ioco

Machiel van der Bijl¹ *, Arend Rensink¹ and Jan Tretmans²

¹ Software Engineering, Department of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
{vdbijl,rensink}@cs.utwente.nl

² Software Technology Research Group, University of Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
tretmans@cs.kun.nl

Abstract. Component based testing concerns the integration of components which have already been tested separately. We show that, with certain restrictions, the **ioco**-test theory for conformance testing is suitable for component based testing, in the sense that the integration of fully conformant components is guaranteed to be correct. As a consequence, there is no need to re-test the integrated system for conformance.

This result is also relevant for *testing in context*, since it implies that every failure of a system embedded in a test context can be reduced to a fault of the system itself.

Contents

1	Introduction	2
2	Formal preliminaries	3
3	Approach	6
3.1	Testing scenario	6
3.2	Component-based testing	7
3.3	Example	8
4	Compositionality for synchronization and hiding	10
4.1	Synchronization	10
4.2	Hiding	11
5	Demonic completion	13
6	Conclusions	17
A	Appendix: proofs	19
A.1	Proofs of section 4.1 (Congruence properties for synchronization)	19
A.2	Proofs of Section 4.2 (Congruence properties for hiding)	24
A.3	Proofs of section 5	33
	References	43

* This research was supported by Ordina Finance and by the dutch research programme PROGRESS under project: TES5417: Atomyste – ATOM splitting in eM-bedded sYSTems TEsting.

1 Introduction

Testing is an important activity in assessing and controlling the quality of systems, in particular in distributed and communicating systems. Ideally, tests are developed from a specification of required behavior, and subsequently executed, be it manually or automatically. The outcome of the test is analyzed leading to a verdict about the correctness of the implementation under test (IUT) with respect to its specification. Test tools may support this testing process.

When a *formal* specification of the system is available, this opens ways toward automatic, specification based test generation. For this purpose several theories of formal testing and test generation, with corresponding tools, have been developed. One of these theories is the so-called **ioco**-test theory, which works on the basis of labeled transition systems [11, 12]. The name **ioco**, which stands for *input/output conformance*, refers to the implementation relation (i.e., notion of correctness) on which the theory and the test generation algorithm have been built. A number of tools implement this algorithm, among which there are TGV [5], TESTGEN [6] and TorX [1].

Two open issues in testing theory in general, and the **ioco**-theory in particular, are *component based testing* — also referred to as compositional testing — and *testing in context*. For instance, for the testing theory based on Finite-State-Machines (FSM) this issue has been studied in [9].

Component based testing considers integration testing of components which have already been tested separately. The question is what can be concluded from the individual tests of the separate components, and what should be (re)tested on the integration or system level. The aim is to determine whether the composition of (correct) component implementations always conforms to the composition of their specifications. In the formal domain, the question is whether the implementation relation is a pre-congruence for composition of components.

Another scenario, with similar characteristics, is *testing in context*. This refers to the situation that a tester cannot directly access the interface of the implementation under test but there is a third system in between, such as a network or protocol unit. This intermediate system is called the *test context* [7, 8, 10]. The tester can only indirectly observe and control the IUT via the test context. This makes testing weaker, in the sense that there are fewer possibilities for observation and control of the IUT.

With testing in context, the question is what can be concluded with respect to conformance of the IUT from testing the composition of IUT and its context: whether faults in the IUT can be detected by testing the composition of IUT and test context, and whether a failure of this composition always indicates a fault of the IUT. This question is the converse of component based testing: when testing in context we wish to detect errors in the IUT — a component — by testing it in composition with the test context, whereas in component based testing we wish to infer correctness of the integrated system from conformance of the individual components. In the formal domain, both boil down to the question whether the implementation relation is a pre-congruence for the composition operators.

This paper studies pre-congruence properties of **ioco** for two operations on labeled transition systems (parallel composition and hiding) that model integration of components — either different parts of the IUT or the IUT and the test context. If **ioco** is a pre-congruence for these operations, it follows that correctness of the parts (the components) implies correctness of the whole (the system), or that a fault in the whole (IUT and test context) implies a fault in the component (IUT).

We show that **ioco** is a pre-congruence in the absence of *implicit underspecification*. One way to satisfy this condition is to only allow specifications which are *input enabled*. Another way is to make the underspecification explicit by *completion*. We show that, in particular, *demonic completion* is suitable for this purpose. This leads to a new implementation relation, baptized **ioco_U**, which is slightly weaker than **ioco**.

Overview. The next section recalls some basic concepts and definitions about transition systems and **ioco**. Section 3 sets the scene and formalizes the problems of component based testing and testing in context. Section 4 studies the pre-congruence properties of **ioco** for parallel composition and hiding. Section 5 discusses underspecification, and approaches to complete specifications with implicit underspecification. Section 6 concludes with some final remarks and an assessment of the results. The proofs of theorems and propositions used in this paper can be found in the appendix.

2 Formal preliminaries

This section recalls the aspects of the theory behind **ioco** that are used for this paper; see [11] for a more detailed exposition.

Labeled Transition Systems. A labeled transition system (LTS) description is defined in terms of states and labeled transitions between states, where the labels indicate what happens during the transition.

Labels are taken from a global set \mathbf{L} . We use a special label $\tau \notin \mathbf{L}$ to denote an internal action. For arbitrary $L \subseteq \mathbf{L}$, we use L_τ as a shorthand for $L \cup \{\tau\}$. We deviate from the standard definition of labeled transition systems in that we assume the label set of an LTS to be partitioned in an input and an output set.

Definition 1. A labeled transition system is a 5-tuple $\langle Q, I, U, T, q_0 \rangle$ where

- Q is a non-empty countable set of states;
- $I \subseteq \mathbf{L}$ is the countable set of input labels;
- $U \subseteq \mathbf{L}$ is the countable set of output labels, which is disjoint from I ;
- $T \subseteq Q \times (I \cup U \cup \{\tau\}) \times Q$ is a set of triples, the transition relation;
- $q_0 \in Q$ is the initial state.

We use L as shorthand for the entire label set ($L = I \cup U$); furthermore, we use Q_p, I_p etc. to denote the components of an LTS p . We commonly write $q \xrightarrow{\lambda} q'$ for

$(q, \lambda, q') \in T$. Since the distinction between inputs and outputs is important, we sometimes use a question mark before a label to denote input and an exclamation mark to denote output. We denote the class of all labeled transition systems over I and U by $\mathcal{LTS}(I, U)$. We represent a labeled transition system in the standard way, by a directed, edge-labeled graph where nodes represent states and edges represent transitions.

A state that cannot do an internal action is called *stable*. A state that cannot do an output or internal action is called *quiescent*. We use the symbol δ ($\notin \mathbf{L}_\tau$) as a pseudo-label to represent quiescence: that is, $p \xrightarrow{\delta} p$ stands for the absence of any transition $p \xrightarrow{\lambda} p'$ with $\lambda \in U_\tau$. For an arbitrary $L \subseteq \mathbf{L}_\tau$, we use L_δ as a shorthand for $L \cup \{\delta\}$.

An LTS is called *strongly responsive* if it always eventually enters a quiescent state; in other words, if it does not have any infinite U_τ -labeled paths. For technical reasons we restrict $\mathcal{LTS}(I, U)$ to strongly responsive transition systems.

A *trace* is a finite sequence of observable actions. The set of all traces over L ($\subseteq \mathbf{L}$) is denoted by L^* , ranged over by σ , 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 use the standard notation with single and double arrows for traces: $q \xrightarrow{a_1 \dots a_n} q$ denotes $q \xrightarrow{a_1} \dots \xrightarrow{a_n} q'$, $q \xRightarrow{\epsilon} q'$ denotes $q \xrightarrow{\tau \dots \tau} q'$ and $q \xRightarrow{a_1 \dots a_n} q$ denotes $q \xRightarrow{\epsilon} \xrightarrow{a_1} \xRightarrow{\epsilon} \dots \xrightarrow{a_n} \xRightarrow{\epsilon} q'$ (where $a_i \in \mathbf{L}_{\tau\delta}$).

We will not always distinguish between a labeled transition system and its initial state: if $p = \langle Q, I, U, T, q_0 \rangle$, then we will identify the process p with its initial state q_0 , and we write, for example, $p \xrightarrow{\sigma} q_1$ instead of $q_0 \xrightarrow{\sigma} q_1$.

Input-output transition systems. An *input-output transition system* (IOTS) is a labeled transition system that is completely specified for input actions: that is, all actions in I are enabled in any state of the transition system.

Definition 2. An input-output transition system $p = \langle Q, I, U, T, q_0 \rangle$ is a labeled transition system for which all inputs are enabled in all states:

$$\forall q \in Q, a \in I : q \xRightarrow{a}$$

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

Composition of labeled transition systems. The integration of components can be modeled algebraically by putting the components in parallel while synchronizing and internalizing their common actions. In process algebra, the synchronization and internalization are typically regarded as two separate operations; the latter is commonly called *hiding*.

Since we have modeled the components as LTS's, these operations translate to constructions over \mathcal{LTS} . Concretely, for two given LTS's p and q and a given set of actions V we define

- The *synchronization* of p and q , denoted $p \parallel q$. This is defined only if the input and output actions of p and q are distinct (in a way made precise

below). The states of the resulting transition system are also denoted $p' \parallel q'$, where p' and q' are states of p and q , respectively.

- The *hiding* in p of the actions in V , denoted **hide** V **in** p . This is defined only if V consists of output actions of p (see below). The states of the resulting transition system are also denoted **hide** V **in** p' , where p' is a state of p . Note that hiding input actions does not make sense.

Note that these constructions are only *partial*: there are constraints on the input and output sets. Moreover, parallel composition may give rise to an LTS that is not strongly responsive, even if the components are. For the time being, we do not try to analyze this but implicitly restrict ourselves to cases where the parallel composition *is* strongly responsive (thus, this is another source of partiality of the construction).

Definition 3. For $i = 1, 2$ let $p_i = \langle Q_i, I_i, U_i, T_i, p_i \rangle$ be a transition system.

- If $I_1 \cap I_2 = U_1 \cap U_2 = \emptyset$ then $p_1 \parallel p_2 =_{\text{def}} \langle Q, I, U, T, p_1 \parallel p_2 \rangle$ where
 - $Q = \{q_1 \parallel q_2 \mid q_1 \in Q_1, q_2 \in Q_2\}$;
 - $I = (I_1 \setminus U_2) \cup (I_2 \setminus U_1)$;
 - $U = U_1 \cup U_2$.
 - T is the minimal set satisfying the following inference rules ($\mu \in L_\tau$):

$$\begin{array}{l} q_1 \xrightarrow{\mu} q'_1, \mu \notin L_2 \quad \vdash q_1 \parallel q_2 \xrightarrow{\mu} q'_1 \parallel q_2 \\ q_2 \xrightarrow{\mu} q'_2, \mu \notin L_1 \quad \vdash q_1 \parallel q_2 \xrightarrow{\mu} q_1 \parallel q'_2 \\ q_1 \xrightarrow{\mu} q'_1, q_2 \xrightarrow{\mu} q'_2, \mu \neq \tau \quad \vdash q_1 \parallel q_2 \xrightarrow{\mu} q'_1 \parallel q'_2 \end{array}$$

- If $V \subseteq U_1$, then **hide** V **in** $p_1 =_{\text{def}} \langle Q, I_1, U_1 \setminus V, T, \mathbf{hide} V \mathbf{in} p_1 \rangle$ where
 - $Q = \{\mathbf{hide} V \mathbf{in} q_1 \mid q_1 \in Q_1\}$;
 - T is the minimal set satisfying the following inference rules ($\mu \in L_\tau$):

$$\begin{array}{l} q_1 \xrightarrow{\mu} q'_1, \mu \notin V \quad \vdash \mathbf{hide} V \mathbf{in} q_1 \xrightarrow{\mu} \mathbf{hide} V \mathbf{in} q'_1 \\ q_1 \xrightarrow{\mu} q'_1, \mu \in V \quad \vdash \mathbf{hide} V \mathbf{in} q_1 \xrightarrow{\tau} \mathbf{hide} V \mathbf{in} q'_1 \end{array}$$

The following proposition states that these constructions yield LTS's, and IOTS's when applied to IOTS's.

Proposition 4. Let $p, q \in \mathcal{LTS}$ with $I_p \cap I_q = U_p \cap U_q = \emptyset$, and let $V \subseteq U_p$.

1. If $p \parallel q$ is strongly responsive then $p \parallel q \in \mathcal{LTS}((I_p \setminus U_q) \cup (I_q \setminus U_p), U_p \cup U_q)$; moreover, $p \parallel q \in \mathcal{IOTS}$ if $p, q \in \mathcal{IOTS}$.
2. **hide** V **in** $p \in \mathcal{LTS}(I_1, U_1 \setminus V)$; moreover, **hide** V **in** $p \in \mathcal{IOTS}$ if $p \in \mathcal{IOTS}$.

Conformance. The testing scenario on which **io** is based assumes that two things are given:

- An LTS constituting a specification of required behavior;
- An implementation under test. We treat the IUT as a black box. In order to reason about it we assume it can be modeled as an IOTS (an IUT is an object in the real world). This assumption is referred to as the test hypothesis [2, 7]. We want to stress that we do not need to *have* this model when testing the IUT. We only *assume* that the implementation *could* be modeled as an IOTS.

Given a specification s and an (assumed) model of the IUT i , the relation $i \mathbf{ioco} s$ expresses that i conforms to s . Whether this holds is decided on the basis of the *suspension traces* of s : it must be the case that, after any such trace σ , every output action (and also quiescence) that i is capable of should be allowed according to s . This is formalized by defining $p \mathbf{after} \sigma$ (the set of states that can be reached in p after the suspension trace σ), $out(p)$ (the set of output and δ -actions of p) and $Straces(p)$ (the suspension traces of p).

Definition 5. Let $p \in \mathcal{LTS}(I, U)$, let $P \subseteq Q_p$ be a set of states in p , let $i \in \mathcal{IOTS}(I, U)$, $s \in \mathcal{LTS}(I, U)$ and let $\sigma \in \mathbf{L}_\delta^*$.

1. $p \mathbf{after} \sigma =_{\text{def}} \{ p' \mid p \xrightarrow{\sigma} p' \}$
2. $out(p) =_{\text{def}} \{ x \in U \mid p \xrightarrow{x} \} \cup \{ \delta \mid p \xrightarrow{\delta} \}$
3. $out(P) =_{\text{def}} \bigcup \{ out(p) \mid p \in P \}$
4. $Straces(p) =_{\text{def}} \{ \sigma \in \mathbf{L}_\delta^* \mid p \xrightarrow{\sigma} \}$

The following defines the implementation relation \mathbf{ioco} , modulo a function \mathcal{F} that generates a set of testable traces from a specification. In this definition 2^X denotes the powerset of X , for an arbitrary set X .

Definition 6. Given a function $\mathcal{F} : \mathcal{LTS}(I, U) \rightarrow 2^{\mathbf{L}_\delta^*}$, we define $\mathbf{ioco}_\mathcal{F} \subseteq \mathcal{IOTS}(I, U) \times \mathcal{LTS}(I, U)$ as follows:

$$i \mathbf{ioco}_\mathcal{F} s \iff \forall \sigma \in \mathcal{F}(s) : out(i \mathbf{after} \sigma) \subseteq out(s \mathbf{after} \sigma)$$

We will use the notation \mathbf{ioco} as the notation for $\mathbf{ioco}_{Straces}$. For more details about \mathbf{ioco} we refer to [11].

3 Approach

To clarify our approach, we briefly recapitulate the testing framework. We focus on the way testing is translated to a formal setting according to the \mathbf{ioco} -theory. We present the results in this section in the context of compositional testing. The consequences for testing in context will be discussed in the final section.

We assume that implementations can be modeled as input-output transition systems. Thus, we require implementations to be input enabled, to reflect the fact that input actions cannot realistically be refused. Specifications are given as general labeled transition systems (with distinguished input and output sets), which means that they are *not* necessarily input enabled. This reflects the idea that a specification may be *partial*: the absence of an input transition from a certain state of the specification means that the behavior after this input is unconstrained — anything goes.

3.1 Testing scenario

We study systems that are obtained by integrating independently implemented (and tested) components, or (in the case of testing in context) an IUT and its test

context. The behavior of such a system is described by the parallel composition of the individual transition systems. Output actions of one component that are in the input label set of another component are *synchronized*, resulting in a single, internal transition of the overall system. Actions of a component that are not in the label set of another component are not synchronized, resulting in a single observable transition of the overall system.

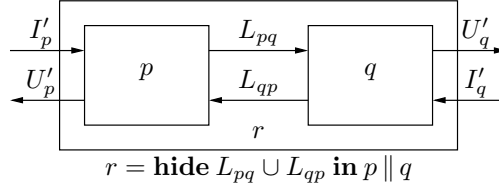


Fig. 1. Parallel composition of components with hiding

This gives rise to the scenario depicted in figure 1, which shows the composition of components p and q , with input and output alphabets I_p and U_p , and I_q and U_q , respectively, such that $I_p \cap I_q = U_p \cap U_q = \emptyset$. The sets $I'_p, U'_p, L_{pq}, L_{qp}, I'_q, U'_q$ are derived from the input/output alphabets as follows:

- $L_{pq} = U_p \cap I_q$: outputs of p that are synchronized with inputs of q ;
- $L_{qp} = U_q \cap I_p$: outputs of q that are synchronized with inputs of p ;
- $I'_p = I_p \setminus U_q$: remaining (i.e., unsynchronized) inputs of p ;
- $U'_p = U_p \setminus I_q$: remaining (i.e., unsynchronized) outputs of p ;
- $I'_q = I_q \setminus U_p$: remaining (i.e., unsynchronized) inputs of q ;
- $U'_q = U_q \setminus I_p$: remaining (i.e., unsynchronized) outputs of q .

Formally, this type of composition is modeled by parallel composition, as embodied in the operator “ \parallel ”, followed by hiding, as embodied in “ $\mathbf{hide} \ V \ \mathbf{in} \ _$ ”, where $V = L_{pq} \cup L_{qp}$ is the set of synchronized actions.

An implicit assumption in this composition scenario is that different components may not have overlapping input or output sets; that is, no action can be an input to two different components or an output from two different components.

3.2 Component-based testing

We now paraphrase the question of component-based testing, discussed in the introduction, as follows: “Given that the components p and q have been tested to be **io-co**-correct (according to their respective specifications), may we conclude that their integration is also **io-co**-correct (according to the integrated specification)?” If the component specifications are LTS’s, the component implementations are modeled by IOTS’s, and their integration by parallel composition

followed by hiding, this boils down to the following questions in our formal framework (where $i_k \in \mathcal{IOTS}(I_k, U_k)$ and $s_k \in \mathcal{LTS}(I_k, U_k)$ for $k = 1, 2$, with $I_1 \cap I_2 = U_1 \cap U_2 = \emptyset$):

Q1: Given $i_k \text{ ioco } s_k$ for $k = 1, 2$, is it the case that $i_1 \parallel i_2 \text{ ioco } s_1 \parallel s_2$?

Q2: Given $i_1 \text{ ioco } s_1$, is it the case that $(\text{hide } V \text{ in } i_1) \text{ ioco } (\text{hide } V \text{ in } s_1)$ for arbitrary $V \subseteq U_1$?

If the answer to both questions is “yes”, then we may conclude the following:

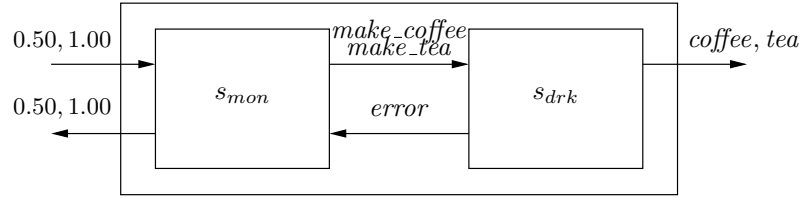
Corollary 7. *If $i_k \in \mathcal{IOTS}(I_k, U_k)$ and $s_k \in \mathcal{LTS}(I_k, U_k)$ for $k = 1, 2$ with $I_1 \cap I_2 = U_1 \cap U_2 = \emptyset$ and $V = (I_1 \cap U_2) \cup (U_1 \cap I_2)$, then*

$$i_1 \text{ ioco } s_1 \wedge i_2 \text{ ioco } s_2 \Rightarrow (\text{hide } V \text{ in } i_1 \parallel i_2) \text{ ioco } (\text{hide } V \text{ in } s_1 \parallel s_2) .$$

This is equivalent to saying that **ioco** is suitable for component based testing.

We study the above congruence questions in the next section. We will show that the answer to Q1 and Q2 in general is *no*. Instead, we can show that the answer to Q1 and Q2 is *yes* if s_1 and s_2 are completely specified.

3.3 Example



$$s_{cof} = \text{hide } \{ \text{make_coffee}, \text{make_tea}, \text{error} \} \text{ in } s_{mon} \parallel s_{drk}$$

$$i_{cof} = \text{hide } \{ \text{make_coffee}, \text{make_tea}, \text{error} \} \text{ in } i_{mon} \parallel i_{drk}$$

Fig. 2. Architecture of coffee machine in components.

To illustrate component based testing, we use two components of a coffee machine: a “money component” that handles the inserted coins and a “drink component” that takes care of preparing and pouring the drinks, see Figure 2.

The money component accepts coins of €1 and of €0.50 as input from the environment. After insertion of a €0.50 coin (respectively €1 coin), the money component orders the drink component to make tea (respectively coffee).

The drink component interfaces with the money component and the environment. If the money component orders it to make tea (respectively coffee) it outputs tea (respectively coffee) to the environment. If anything goes wrong in the drink making process, the component gives an error signal.

The coffee machine is the parallel composition of the money component and the drink component, in which the “make coffee” command, the “make tea” command and the “error” signal are hidden. One can think of the parallel composition as establishing the connection between the money component and the drink component, whereas hiding means that the communication between the components is not observable anymore; only communication with the environment can be observed.

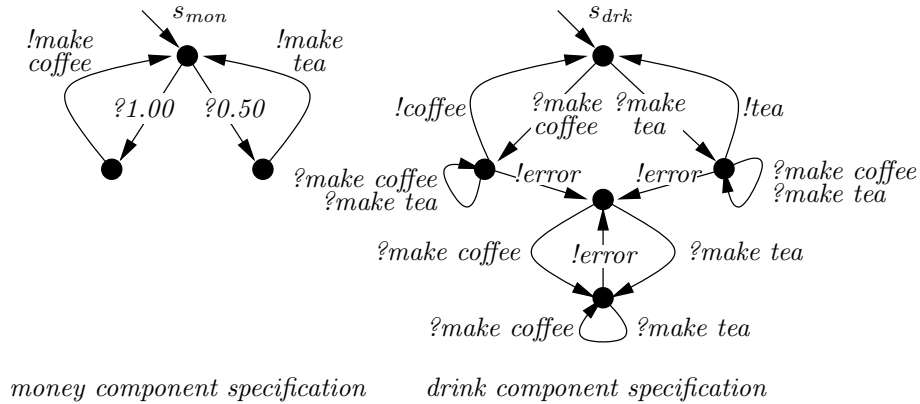


Fig. 3. Specification of money and drink components as LTS's.

Models In Figure 3 we show the behavioral specification of the components as LTS's. Note that the money component is underspecified for the *error* input label and that the drink component cannot recover from an error state, and while in the error state it cannot produce tea or coffee. Figure 4 shows implementation models of the “money component”, i_{mon} , and the “drink component”, i_{drk} . We have used transitions labeled with “?” as an abbreviation for all the non-specified input actions from the alphabet of the component. The money component has input label set, $I_{mon} = \{0.50, 1.00, error\}$, output label set $U_{mon} = \{make_coffee, make_tea, 0.50, 1.00\}$; $s_{mon} \in \mathcal{LTS}(I_{mon}, U_{mon})$, $i_{mon} \in \mathcal{IOTS}(I_{mon}, U_{mon})$. For the drink component $I_{drk} = \{make_coffee, make_tea\}$ and $U_{drk} = \{coffee, tea, error\}$ are the input and output label sets; $s_{drk} \in \mathcal{LTS}(I_{drk}, U_{drk})$, $i_{drk} \in \mathcal{IOTS}(I_{drk}, U_{drk})$.

In the implementations of the components we choose to improve upon the specification, by adding functionality. This is possible since, as we have discussed, the specification may in places be under-specified. The extra functionality of i_{mon} compared to its specification s_{mon} is that it can handle error signals: it reacts by returning €1.00. i_{drk} is also changed with respect to its specification s_{drk} : making tea never produces an error signal. Since implementations are input

enabled, we have chosen that all non specified inputs are ignored, i.e., the system remains in the same state.

We have $i_{mon} \mathbf{ioco} s_{mon}$ and $i_{drk} \mathbf{ioco} s_{drk}$. The question now is whether the integrated specifications and implementations, as given by s_{cof} and i_{cof} in Figure 2, are also \mathbf{ioco} correct. We discuss this in the next section, to illustrate the compositionality properties discussed there.

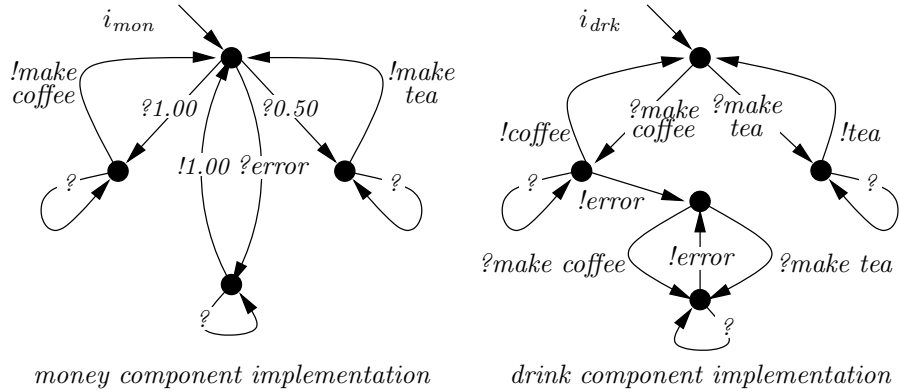


Fig. 4. Implementation of the money and drink components as IOTS's.

4 Compositionality for synchronization and hiding

In this section we address the questions Q1 and Q2 formulated above (Section 3.2), using the coffee machine example to illustrate our results.

4.1 Synchronization

The property that we investigate for parallel composition is: *if* we have two correct component implementations according to \mathbf{ioco} , *then* the implementation remains correct after synchronizing the components. It turns out that in general this property does not hold, as we show in the following example.

Example 8. Regard the LTS's in figure 5. On the left hand side we show the specifications and on the right hand side the corresponding implementations. The models have the following label sets: $s_1 \in \mathcal{LTS}(\{x\}, \emptyset)$, $i_1 \in \mathcal{IOTS}(\{x\}, \emptyset)$, $s_2 \in \mathcal{LTS}(\emptyset, \{x\})$, $i_2 \in \mathcal{IOTS}(\emptyset, \{x\})$. The suspension traces of s_1 are given by $\delta^* \cup \delta^*?x\delta^*$ and the suspension traces of s_2 are given by $\{\epsilon, !x\} \cup !x!x\delta^*$. We have $i_1 \mathbf{ioco} s_1$ and $i_2 \mathbf{ioco} s_2$.

After we take the parallel composition of the two specifications we get $s_1 \parallel s_2$, see figure 5 (the corresponding implementation is $i_1 \parallel i_2$). From this specification the suspension traces are: $\{\epsilon\} \cup !x\delta^*$. We see that $out(i_1 \parallel i_2 \text{ after } !x) = \{!x\} \not\subseteq out(s_1 \parallel s_2 \text{ after } !x) = \{\delta\}$; in other words: $i_1 \parallel i_2 \mathbf{ioco} s_1 \parallel s_2$.

Analysis shows that $i_1 \mathbf{ioco} s_1$, because \mathbf{ioco} allows underspecification of input actions. However, the semantics of the parallel composition operator does not take underspecification of input actions into account. Although s_2 can output a second x , it cannot do so in $s_1 \parallel s_2$, because s_1 cannot input the second x . We say that s_1 is underspecified for the second x , because although an implementation is allowed to accept a second x as input, the correct behavior after the second x is not specified. In other words the specification is underspecified for the second x as input.

It turns out that if we forbid underspecification, i.e., if the specification explicitly prescribes for any possible input what the allowed responses are, then we do not have this problem. In fact in that case we have the desired compositionality property. This property is expressed in the following theorem. For a proof see the appendix.

Theorem 9. *Let $s_1, i_1 \in IOTS(I_1, U_1)$, $s_2, i_2 \in IOTS(I_2, U_2)$, with $I_1 \cap I_2 = U_1 \cap U_2 = \emptyset$.*

$$i_1 \mathbf{ioco} s_1 \wedge i_2 \mathbf{ioco} s_2 \Rightarrow i_1 \parallel i_2 \mathbf{ioco} s_1 \parallel s_2$$

Our running example (Section 3.3) shows the same problem illustrated in example 8. Although the implementations of the money component and the drink component are \mathbf{ioco} correct with respect to their specifications, it turns out that the parallel composition of i_{mon} and i_{drk} is not:

$$\begin{aligned} out(i_{mon} \parallel i_{drk} \text{ after } ?1.00.!make_coffee) &= \{!coffee, !error\} \\ out(s_{mon} \parallel s_{drk} \text{ after } ?1.00.!make_coffee) &= \{!coffee\} \end{aligned}$$

Note that the internal signals are still visible as output actions. To turn them into internal actions is the task of the *hiding* operator, discussed below.

4.2 Hiding

The property that we investigate for hiding is the following: *if* we have a correct implementation according to \mathbf{ioco} , *then* the implementation remains correct af-

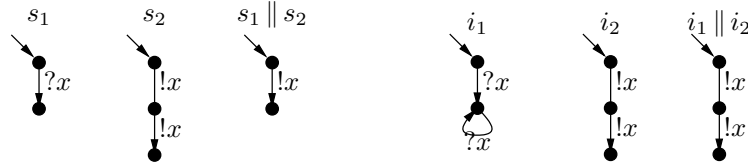


Fig. 5. Counter example to compositionality for parallel composition; see Example 8.

ter hiding (some of the) output actions. It turns out that, as for synchronization, in general this property does not hold.

Example 10. Consider the implementation i and specification s in Figure 6, both with input set $\{a\}$ and output set $\{x, y\}$. $\{\epsilon\} \cup ?a\delta^* \cup !x\delta^*$ are the suspension traces of s . We see that i **io**co s .

After hiding the output action x , we get the specification **hide** x **in** s , and implementation **hide** x **in** i ; now the specification has suspension traces $?a\delta^* \cup \delta^*$. We have $out(\mathbf{hide} \ x \ \mathbf{in} \ i \ \mathbf{after} \ a) = \{\delta, y\} \not\subseteq out(\mathbf{hide} \ x \ \mathbf{in} \ s \ \mathbf{after} \ a) = \{\delta\}$; in other words, $(\mathbf{hide} \ x \ \mathbf{in} \ i)$ **io**co $(\mathbf{hide} \ x \ \mathbf{in} \ s)$.

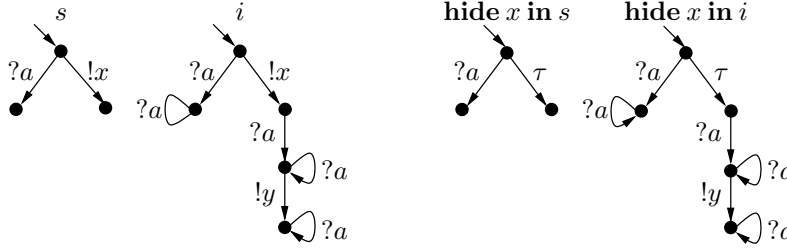


Fig. 6. Counter-example to compositionality for hiding; see Example 10.

An analysis of the above example shows that s was underspecified, in the sense that it fails to prescribe how an implementation should behave after the trace $!x?a$. The proposed implementation i uses the implementation freedom by having an unspecified y -output after $!x?a$. However, if x becomes unobservable due to hiding, then the traces $!x?a$ and $?a$ collapse and become indistinguishable: in **hide** x **in** s and **hide** x **in** i they both masquerade as the trace $?a$. Now **hide** x **in** s appears to specify that after $?a$, only quiescence (δ) is allowed; however, **hide** x **in** i still has this unspecified y -output. In other words, hiding creates confusion about what part of the system is underspecified.

It follows that if we rule out underspecification, i.e., we limit ourselves to specifications that are IOTS's then this problem disappears. In fact, in that case we do have the desired congruence property. This is stated in the following theorem.

Theorem 11. *If $i, s \in \mathcal{IOTS}(I, U)$ with $V \subseteq U$, then:*

$$i \text{ io}co s \Rightarrow (\mathbf{hide} \ V \ \mathbf{in} \ i) \text{ io}co (\mathbf{hide} \ V \ \mathbf{in} \ s)$$

Using our running example, we show that the problem illustrated by Example 10 is not unrealistic. The integrated coffee system implementation i_{cof} , consisting of the money and drink component implementations, after hiding the

synchronized actions ($make_coffee, make_tea, error$), can do the trace $?1.00!1.00$ (ejecting $\text{€}1.00$ after a *hidden* error). Hence $!1.00 \in out(i_{cof} \mathbf{after} ?1.00)$ whereas $!1.00 \notin out(s_{cof} \mathbf{after} ?1.00)$ (where $?1.00 \in Straces(s_{cof})$) (see Figure 2 for s_{cof}, i_{cof}).

So we can conclude that although the component implementations are **io** correct with respect to their specifications, the integrated system is not.

5 Demonic completion

We have shown in the previous section that **io** is a pre-congruence for parallel composition and hiding when restricted to $IOTS \times IOTS$. However, in the original theory [11] $\mathbf{io} \subseteq IOTS \times LTS$; the specifications are LTS's. The intuition behind this is that **io** allows underspecification of input actions. In this section we present a function that transforms LTS's into IOTS's in a way that complies with this notion of underspecification. We will show that this leads to a new implementation relation that is slightly weaker than **io**.

Underspecification comes in two flavors: underspecification of input actions and underspecification of output actions. Underspecification of output actions is always explicit; in an LTS it is represented by a non-deterministic choice between several output actions. The intuition behind this is that we do not know or care which of the output actions is implemented, as long as at least one is. Underspecification of input actions is always implicit; it is represented by absence of the respective input action in the LTS. The intuition behind underspecification of input actions is that after an unspecified input action we do not know or care what the behavior of the specified system is. This means that in an underspecified state — i.e., a state reached after an unspecified input action — every action from the label set is correct, including quiescence. Following [3] we call this kind of behavior *chaotic*.

In translating LTS's to IOTS's, we propose to model underspecification of input actions explicitly. Firstly, we model chaotic behavior through a state q_χ (where χ stands for chaos) with the property: $\forall \lambda \in U : q_\chi \xrightarrow{\lambda} q_\chi$ and $\forall \lambda \in I : q_\chi \xrightarrow{\delta^* \cdot \lambda} q_\chi$. Secondly, we add for every stable state q (of a given LTS) that is underspecified for an input a a transition (q, a, q_χ) . This turns the LTS into an IOTS. After [4] we call this procedure *demonic* completion — as opposed to *angelic* completion, where unspecified inputs are discarded (modeled by adding self-loop transitions).

Definition 12 (demonic completion). $\Xi : LTS(I, U) \rightarrow IOTS(I, U)$ is defined by $\langle Q, I, U, T, q_0 \rangle \mapsto \langle Q', I, U, T', q_0 \rangle$, where

$$\begin{aligned} Q' &= Q \cup \{q_\chi, q_\Omega, q_\Delta\}, \text{ where } q_\chi, q_\Omega, q_\Delta \notin Q \\ T' &= T \cup \{(q, a, q_\chi) \mid q \in Q, a \in I, q \xrightarrow{a} \cdot, q \xrightarrow{\tau} \cdot\} \\ &\quad \cup \{(q_\chi, \tau, q_\Omega), (q_\chi, \tau, q_\Delta)\} \cup \{(q_\Omega, \lambda, q_\chi) \mid \lambda \in L\} \cup \{(q_\Delta, \lambda, q_\chi) \mid \lambda \in I\} \end{aligned}$$

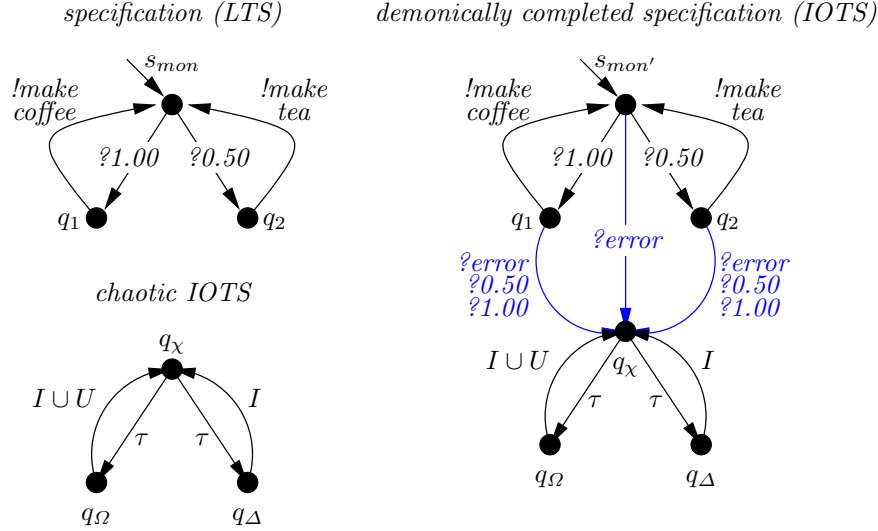


Fig. 7. Demonic completion of an LTS specification.

Example 13. To illustrate the demonic completion of implicit underspecification, we use the money component of section 3.3. The LTS specification of the money component is given in the top left corner of Figure 7. The IOTS that models our chaos property is given in the bottom left corner. For every stable state of the specification, the function Ξ adds an arrow for every unspecified input, from that state to state q_χ . For example, every state is underspecified for input action *error*, so we add a transition from every state to q_χ for *error*. The states q_1 and q_2 are underspecified for 0.50 and 1.00, so we add transitions for these inputs from q_1 and q_2 to q_χ . The resulting demonically completed specification is given on the right hand side of Figure 7.

The chaotic IOTS can do every trace that an LTS can do. However it is easy to see that the chaotic IOTS cannot do every imaginable trace. For example it cannot do $\delta \cdot \lambda$, where $\lambda \in U$. We need the following property to show that the chaotic IOTS can do every (sub)trace that an arbitrary LTS can do. The regular set $(U \cup \delta^* I)^* \delta^*$ contains all the elements of L_δ^* except those that contain the trace δ followed by an output. By definition this is of course an impossible trace for an LTS.

Proposition 14. *Let $s \in \mathcal{LTS}(I, U)$.*

$$\forall \sigma \in \text{Straces}(s) : \sigma \in (U \cup \delta^* I)^* \delta^*$$

An important property of demonic completion is that it only adds transitions from *stable* states with underspecified inputs in the original LTS to q_χ . More-

over, it does not delete states or transitions. This is expressed in the following proposition.

Proposition 15. *Let $s \in \mathcal{LTS}(I, U)$.*

$$\forall \sigma \in L_\delta^*, q' \in Q_s : s \xrightarrow{\sigma} q' \Leftrightarrow \Xi(s) \xrightarrow{\sigma} q'$$

Furthermore, the chaotic IOTS acts as a kind of sink: once one of the added states (q_χ , q_Ω or q_Δ) has been reached, they will never be left anymore. Together with proposition 14 it is easy to see that q_χ can do all the (sub)traces that an arbitrary LTS can do.

Proposition 16. *Let $s \in \mathcal{LTS}(I, U)$ and q_χ be the chaotic state in the demonically completed LTS $\Xi(s)$.*

$$\forall \sigma \in (U \cup \delta^* I)^* \delta^* : q_\chi \xrightarrow{\sigma} q' \wedge q' \notin Q_s$$

We use the notation “ $\mathbf{ioco} \circ \Xi$ ” to denote that before applying \mathbf{ioco} , the LTS specification is transformed to an IOTS by Ξ ; i.e., $i(\mathbf{ioco} \circ \Xi)s \Leftrightarrow i \mathbf{ioco} \Xi(s)$.

Theorem 17. $\mathbf{ioco} \subseteq \mathbf{ioco} \circ \Xi$

Note that the opposite is not true, i.e., $i(\mathbf{ioco} \circ \Xi)s \not\subseteq i \mathbf{ioco} s$. Figure 8 shows that the counter-example for the pre-congruence of hiding is also a counter-example for the reverse direction of this proposition. On the left hand side we see the specification s , next to it is the demonically completed specification $\Xi(s)$ and on the right is the implementation. Now we have that $i(\mathbf{ioco} \circ \Xi)s$ and $i \mathbf{ioco} s$. Thus, $\mathbf{ioco} \circ \Xi$ is a strictly weaker relation than \mathbf{ioco} . We want to stress two aspects of this observation:

- Our investigation, which started with component-based testing, has led to a new variation in the notion of conformance that is different from the relation implemented in, e.g., the TORX tool;
- This new variation considers all previously tested, \mathbf{ioco} -conformant systems correct, but may consider a system correct that was rejected by \mathbf{ioco} . Thus, the variation is relatively harmless.

These properties are a consequence of our choice of the demonic completion function. Other forms of completion, such as angelic completion, result in variants of \mathbf{ioco} which are incomparable to the original relation (i.e., neither weaker nor stronger).

The difference between \mathbf{ioco} and $\mathbf{ioco} \circ \Xi$ occurs when there is nondeterminism in the underspecification, i.e. when the specification can nondeterministically go to two or more different states. In one state the input is specified, whereas in another state the input is underspecified. For example, in Figure 8, s can nondeterministically take the τ -transition. Before the τ -transition, input $?a$ is specified, while after the τ -transition input $?a$ is underspecified. In case of nondeterminism in the underspecification, \mathbf{ioco} takes the specified side, while $\mathbf{ioco} \circ \Xi$ takes the underspecified side.

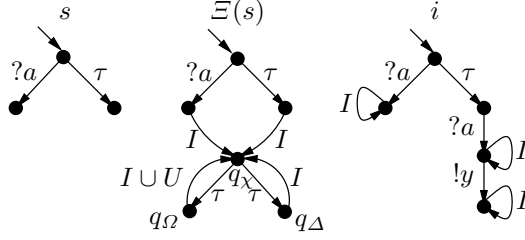


Fig. 8. $i(\mathbf{ioco} \circ \Xi)s \not\equiv i \mathbf{ioco} s$.

Testing The testing scenario is now such that an integrated system can be tested by comparing the individual components to their *demonically completed* specifications. If the components conform, then the composition of implementations also conforms to the composition of the demonically completed specifications.

Corollary 18. *Let $s_1, s_2 \in \mathcal{LTS}(I, U)$ and $i_1, i_2 \in \mathcal{IOTS}(I, U)$*

$$i_1 \mathbf{ioco} \Xi(s_1) \wedge i_2 \mathbf{ioco} \Xi(s_2) \Rightarrow i_1 \parallel i_2 \mathbf{ioco} \Xi(s_1) \parallel \Xi(s_2)$$

Test restriction A disadvantage of demonic completion is that it destroys information about underspecified behavior. On the basis of the underspecified LTS, one can conclude that traces including an unspecified input need not be tested because every implementation will always pass; after completion, however, this is no longer visible, and so automatic test generation will yield many spurious tests.

In order to avoid this, we characterize $\mathbf{ioco} \circ \Xi$ directly over LTS's. In other words, we extend the relation from $\mathcal{IOTS} \times \mathcal{IOTS}$ to $\mathcal{IOTS} \times \mathcal{LTS}$, in such a way as to obtain the same testing power but avoid these spurious tests. For this purpose, we restrict the number of traces after which we test.

Definition 19. *Let $s \in \mathcal{LTS}(I, U)$.*

$$Utraces(s) =_{\text{def}} \{ \sigma \in L_\delta^* \mid s \xrightarrow{\sigma} \wedge (\nexists q', \sigma_1 \cdot a \cdot \sigma_2 = \sigma : a \in I \wedge s \xrightarrow{\sigma_1} q' \wedge q' \xrightarrow{a}) \}$$

Intuitively, the *Utraces* are the *Straces* without the underspecified traces. A trace σ is underspecified if there exists a prefix $\sigma_1 \cdot a$ of σ , with $a \in I$, for which $s \xrightarrow{\sigma_1} q'$ and $q' \xrightarrow{a}$. The following proposition puts it differently: *Utraces*(s) are the *Straces*(s) for which the demonically completed specification, $\Xi(s)$, never leads to a chaotic state.

Proposition 20. *Let $s = \langle Q, I, U, T, q_0 \rangle$ be an arbitrary LTS.*

$$Utraces(s) = \{ \sigma \in Straces(s) \mid (\Xi(s) \text{ after } \sigma) \subseteq Q \}$$

We use \mathbf{ioco}_U as a shorthand for $\mathbf{ioco}_{Utraces}$. In the following proposition we state that \mathbf{ioco}_U is equivalent to $\mathbf{ioco} \circ \Xi$.

Theorem 21. $\mathbf{ioco}_U = \mathbf{ioco} \circ \Xi$

This equivalence is quite intuitive. $\mathbf{ioco} \circ \Xi$ uses fresh states to handle underspecified behavior, which are constructed so as to display chaotic behavior. If $\Xi(s)$ reaches such a state, then all behavior is considered correct. \mathbf{ioco}_U , on the other hand, circumvents underspecified behavior, because it uses *Utraces*, which are *Straces* from which all underspecified traces have been removed.

6 Conclusions

The results of this paper imply that \mathbf{ioco} can be used for component based testing: if components are tested separately and then integrated, the resulting system will also conform to the integrated specification. The integration of components is modeled by the successive application of two algebraic operations: *parallel composition* followed by *hiding*. However, as we have shown, there are some constraints to this scenario. Summarizing the results of this paper:

- We have shown that component-based \mathbf{ioco} -testing works smoothly if the component specifications are input-enabled (i.e., given as input-output transition systems); see theorems 9 and 11.
- We have shown that if the component specifications are *not* input enabled, both parallel composition and hiding are non-compositional (see example 8 and 10): the integrated system may not conform to the integrated specification in spite of the fact that the individual components are fully conformant.
- A solution to this is to *complete* the component specifications before using them for testing purposes; this involves adding input transitions where they are missing. Completion turns the specifications into IOTS's, so that the above result becomes applicable. Completion can be done in many different ways; we have shown that *demonic completion* is consistent with the existing \mathbf{ioco} theory (see theorem 17).
- Testing after completion is in principle (much) more expensive since, due to the nature of IOTS's, even the completion of a finite specification already displays infinite testable behavior. As a final result of this paper, we have shown how to use the original component specifications for testing, *before* completion (see theorem 21). Note that the correctness of the *integrated* system is still only guaranteed with respect to the *completed* component specifications; thus, completion is still an unavoidable step.
- The result of this paper is also relevant for *testing in context*, since it implies that every failure of a system embedded in a test context can be reduced to a fault of the system itself.

Interpretation. The insights gained from these results can be recast in terms of *underspecification*. \mathbf{ioco} recognizes two kinds of underspecification: omitting

input actions from a state (which implies a *don't care* if an input does occur) and including multiple output actions from a state (which allows the implementation to choose between them). It turns out that the first of these two is not compatible with parallel composition and hiding. To make it compatible we have had to modify the theory at two points:

- The underlying set of traces for **io**co has been redefined, resulting in a slightly weaker notion of conformance; previously conformant implementations are still conformant, but it might be that previously non-conformant implementations are allowed under the modified notion of conformance.
- The integration (i.e., parallel composition and hiding) of component specifications has been moved to the domain of IOTS's. Thus, we no longer consider these as constructions on general labeled transition systems; instead, component specifications have to be *completed* prior to their integration.

Testing in context We have discussed the pre-congruence properties mainly in the context of component based testing, but the results can easily be transposed to testing in context. Suppose an implementation under test i is tested via a context c . The tester interacts with c , and c interacts with i ; the tester cannot directly interact with i . Then we have $I_i \subseteq U_c$ and $U_i \subseteq I_c$, and L_i is not observable for the tester, i.e., hidden. The tester observes the system as an implementation in a context:

$$\mathcal{C}[i] = \mathbf{hide} \ I_i \cup U_i \ \mathbf{in} \ c \parallel i$$

Now theorem 9 and 11 directly lead to the following corollary for testing in context.

Corollary 22. *Let $s, i \in \mathcal{IOTS}$ occur in test context $\mathcal{C}[-]$*

$$\mathcal{C}[i] \ \mathbf{io}\phi\mathbf{o} \ \mathcal{C}[s] \Rightarrow i \ \mathbf{io}\phi\mathbf{o} \ s$$

Hence, an error detected while testing the implementation in its context is a real error of the implementation, but not the other way around: an error in the implementation may not be detectable when tested in a context.

Future work. The current state of affairs is not yet completely satisfactory, because the notion of composition that we require is not defined on general labeled transition systems but just on IOTS's. Testing against IOTS's is inferior, in that these models do not allow the “input underspecification” discussed above: for that reason, testing against an IOTS cannot take advantage of information about “don't care” inputs (essentially, *no* testing is required after a “don't care” input, since by definition every behavior is allowed).

We intend to solve this issue by extending IOTS's with a predicate that makes the notion of input underspecification explicit, by designating some states as *chaotic*. Testing can stop when the specification has reached a chaotic state.

A Appendix: proofs

A.1 Proofs of section 4.1 (Congruence properties for synchronization)

Proposition A.1. *Let $p \in \mathcal{IOTS}(I_p, U_p)$, $q \in \mathcal{IOTS}(I_q, U_q)$, $r \in Q_{p \parallel q}$ with $I_p \cap I_q = U_p \cap U_q = \emptyset$.*

1. $\forall a \in L_p \setminus L_q : p \parallel q \xrightarrow{a} r \Leftrightarrow \exists p' : p \xrightarrow{a} p' \wedge r = p' \parallel q$
2. $\forall a \in L_q \setminus L_p : p \parallel q \xrightarrow{a} r \Leftrightarrow \exists q' : q \xrightarrow{a} q' \wedge r = p \parallel q'$
3. $p \parallel q \xrightarrow{\tau} r \Leftrightarrow (\exists p' : p \xrightarrow{\tau} p' \wedge r = p' \parallel q) \vee (\exists q' : q \xrightarrow{\tau} q' \wedge r = p \parallel q')$
4. $\forall a \in (L_p \cap L_q) \cup \{\delta\} : p \parallel q \xrightarrow{a} r \Leftrightarrow \exists p', q' : p \xrightarrow{a} p' \wedge q \xrightarrow{a} q' \wedge r = p' \parallel q'$

Proof.

1. Assume $a \in L_p \setminus L_q$

only if:

$$\begin{aligned} & p \parallel q \xrightarrow{a} r \\ \Rightarrow & (*\text{definition } \parallel *) \\ & \exists p' : p \xrightarrow{a} p' \wedge r = p' \parallel q \end{aligned}$$

if:

$$\begin{aligned} & \exists p' : p \xrightarrow{a} p' \wedge r = p' \parallel q \\ \Rightarrow & (*\text{definition } \parallel *) \\ & p \parallel q \xrightarrow{a} r \end{aligned}$$

2. Analogous to 1
3. Analogous to 1
4. The case $a \in L_p \cap L_q$ is analogous to 1. Here the proof for $a = \delta$ is given.

only if: First we show $r = p \parallel q$, then the remaining part of the proof is given.

$$\begin{aligned} & p \parallel q \xrightarrow{\delta} r \\ \Rightarrow & (*\text{Definition } \delta *) \\ & p \parallel q \xrightarrow{\delta} r \wedge r = p \parallel q \end{aligned} \tag{1}$$

$$\begin{aligned} & p \parallel q \xrightarrow{\delta} r \\ \Rightarrow & (*\text{Definition } \delta *) \\ & \forall \mu \in U_p \cup U_q \cup \{\tau\} : p \parallel q \not\xrightarrow{\mu} \\ \Rightarrow & (*\text{Definition } \parallel *) \\ & \forall \mu \in (U_p \setminus L_q) \cup \{\tau\} : p \not\xrightarrow{\mu} \\ & \wedge \forall \mu \in (U_q \setminus L_p) \cup \{\tau\} : q \not\xrightarrow{\mu} \\ & \wedge \forall \mu \in L_p \cap L_q : p \not\xrightarrow{\mu} \vee q \not\xrightarrow{\mu} \\ \Rightarrow & (*p, q \in \mathcal{IOTS} \text{ (note that } \forall \mu \in I_p : p \not\xrightarrow{\mu} \wedge \forall \mu \in I_q : q \not\xrightarrow{\mu} \text{)}) *) \\ & \forall \mu \in U_p \cup \{\tau\} : p \not\xrightarrow{\mu} \wedge \forall \mu \in U_q \cup \{\tau\} : q \not\xrightarrow{\mu} \\ \Rightarrow & (*\text{definition } \delta \text{ and } r = p \parallel q \text{ (see 1) } *) \\ & p \xrightarrow{\delta} p \wedge q \xrightarrow{\delta} q \wedge r = p \parallel q \end{aligned}$$

$$\begin{aligned}
\text{if: Let } r &= p \parallel q \\
& p \xrightarrow{\delta} p \wedge q \xrightarrow{\delta} q \\
& \Rightarrow (* \text{definition } \delta \text{ } *) \\
& \forall \mu \in U_p \cup \{\tau\} : p \not\xrightarrow{\mu} \wedge \forall \mu \in U_q \cup \{\tau\} : q \not\xrightarrow{\mu} \\
& \Rightarrow (* \text{definition } \parallel \text{ } *) \\
& \forall \mu \in U_p \cup U_q \cup \{\tau\} : p \parallel q \not\xrightarrow{\mu} \\
& \Rightarrow (* \text{definition } \delta \text{ } *) \\
& p \parallel q \xrightarrow{\delta} p \parallel q \\
& \Rightarrow (* \text{Given: } r = p \parallel q \text{ } *) \\
& p \parallel q \xrightarrow{\delta} r
\end{aligned}$$

We now introduce some notation for the projection of a trace on a label set.

Definition A.2. Let $a \in L_\delta$, $\sigma \in L_\delta^*$, $S \subseteq L_\delta$:

$$\begin{aligned}
\epsilon \upharpoonright S &= \epsilon \\
(a \cdot \sigma) \upharpoonright S &= \begin{cases} \sigma \upharpoonright S & a \notin S \\ a \cdot (\sigma \upharpoonright S) & a \in S \end{cases}
\end{aligned}$$

To improve readability we use L_p^δ to denote $L_p \cup \{\delta\}$.

Proposition A.3. Let $p \in \mathcal{IOTS}(I_p, U_p)$, $q \in \mathcal{IOTS}(I_q, U_q)$, $r \in \mathcal{Q}_{p \parallel q}$, $\sigma \in L_\delta^*$, with $L = I_p \cup I_q \cup U_p \cup U_q$ and $I_p \cap I_q = U_p \cap U_q = \emptyset$.

$$p \parallel q \xrightarrow{\sigma} r \Leftrightarrow \exists p', q' : p \xrightarrow{\sigma \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\sigma \upharpoonright L_q^\delta} q' \wedge r = p' \parallel q'$$

Proof.

Only if: Proof by induction on the structure of σ .

Basic step: $\sigma = \epsilon$.

$$\begin{aligned}
& p \parallel q \xrightarrow{\epsilon} r \\
& \Rightarrow (* \text{Proposition A.1.3 and definition of } \xrightarrow{\epsilon} \text{ } *) \\
& \quad \exists p', q' : p \xrightarrow{\epsilon} p' \wedge q \xrightarrow{\epsilon} q' \wedge r = p' \parallel q' \\
& \Rightarrow (* \text{Definition A.2 of projection } *) \\
& \quad \exists p', q' : p \xrightarrow{\epsilon \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\epsilon \upharpoonright L_q^\delta} q' \wedge r = p' \parallel q'
\end{aligned}$$

Induction step: We make the assumption that the proposition holds for σ' in $\sigma = a \cdot \sigma'$, with $a \in L_\delta$. We identify three cases:

1. Assume $a \in L_p^\delta \cap L_q^\delta (= (L_p \cap L_q) \cup \{\delta\})$.

$$\begin{aligned}
& p \parallel q \xrightarrow{a \cdot \sigma'} r \\
\Rightarrow & (* \text{Definition } \xrightarrow{a \cdot \sigma'} *) \\
& \exists r_1, r_2 : p \parallel q \xrightarrow{\epsilon} r_1 \xrightarrow{a} r_2 \xrightarrow{\sigma'} r \\
\Rightarrow & (* \text{Basic step } *) \\
& \exists p_1, q_1, r_2 : p \xrightarrow{\epsilon} p_1 \wedge q \xrightarrow{\epsilon} q_1 \wedge p_1 \parallel q_1 \xrightarrow{a} r_2 \xrightarrow{\sigma'} r \\
\Rightarrow & (* \text{Proposition A.1.4 } *) \\
& \exists p_1, p_2, q_1, q_2 : p \xrightarrow{\epsilon} p_1 \wedge q \xrightarrow{\epsilon} q_1 \wedge p_1 \xrightarrow{a} p_2 \wedge q_1 \xrightarrow{a} q_2 \wedge p_2 \parallel q_2 \xrightarrow{\sigma'} r \\
\Rightarrow & (* \text{Definition } \xrightarrow{a} *) \\
& \exists p_2, q_2 : p \xrightarrow{a} p_2 \wedge q \xrightarrow{a} q_2 \wedge p_2 \parallel q_2 \xrightarrow{\sigma'} r \\
\Rightarrow & (* \text{Induction } *) \\
& \exists p', p_2, q', q_2 : p \xrightarrow{a} p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{a} q_2 \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \wedge r = p' \parallel q' \\
\Rightarrow & (* \text{Definition A.2 of projection } *) \\
& \exists p', q' : p \xrightarrow{(a \cdot \sigma') \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{(a \cdot \sigma') \upharpoonright L_q^\delta} q' \wedge r = p' \parallel q' \\
\Rightarrow & (* \sigma = a \cdot \sigma' *) \\
& \exists p', q' : p \xrightarrow{\sigma \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\sigma \upharpoonright L_q^\delta} q' \wedge r = p' \parallel q'
\end{aligned}$$

2. Assume $a \in L_p^\delta \setminus L_q^\delta (= L_p \setminus L_q)$.

$$\begin{aligned}
& p \parallel q \xrightarrow{a \cdot \sigma'} r \\
\Rightarrow & (* \text{Definition } \xrightarrow{\sigma'} *) \\
& \exists r_1, r_2 : p \parallel q \xrightarrow{\epsilon} r_1 \xrightarrow{a} r_2 \xrightarrow{\sigma'} r \\
\Rightarrow & (* \text{Basic step } *) \\
& \exists p_1, q_1, r_2 : p \xrightarrow{\epsilon} p_1 \wedge q \xrightarrow{\epsilon} q_1 \wedge p_1 \parallel q_1 \xrightarrow{a} r_2 \xrightarrow{\sigma'} r \\
\Rightarrow & (* \text{Proposition A.1.1 } *) \\
& \exists p_1, p_2, q_1 : p \xrightarrow{\epsilon} p_1 \wedge q \xrightarrow{\epsilon} q_1 \wedge p_1 \xrightarrow{a} p_2 \wedge p_2 \parallel q_1 \xrightarrow{\sigma'} r \\
\Rightarrow & (* \text{Definition of } \xrightarrow{a} *) \\
& \exists p_2, q_1 : p \xrightarrow{a} p_2 \wedge q \xrightarrow{\epsilon} q_1 \wedge p_2 \parallel q_1 \xrightarrow{\sigma'} r \\
\Rightarrow & (* \text{Induction } *) \\
& \exists p_2, p', q_1, q' : p \xrightarrow{a} p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\epsilon} q_1 \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \wedge r = p' \parallel q' \\
\Rightarrow & (* \text{Definition A.2 of projection } *) \\
& \exists p', q' : p \xrightarrow{(a \cdot \sigma') \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{(a \cdot \sigma') \upharpoonright L_q^\delta} q' \wedge r = p' \parallel q' \\
\Rightarrow & (* \sigma = a \cdot \sigma' *) \\
& \exists p', q' : p \xrightarrow{\sigma \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\sigma \upharpoonright L_q^\delta} q' \wedge r = p' \parallel q'
\end{aligned}$$

3. Assume $a \in L_q^\delta \setminus L_p^\delta$. This is symmetric with the previous case.

if: By induction on the structure of σ .

Basic step: $\sigma = \epsilon$.

$$\begin{aligned}
& \exists p', q' : p \xrightarrow{\epsilon \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\epsilon \upharpoonright L_q^\delta} q' \\
\Rightarrow & (* \text{Definition A.2 of projection } *) \\
& \exists p', q' : p \xrightarrow{\epsilon} p' \wedge q \xrightarrow{\epsilon} q' \\
\Rightarrow & (* \text{Proposition A.1.3 and definition of } \xrightarrow{\epsilon} *) \\
& \exists p', q' : p \parallel q \xrightarrow{\epsilon} p' \parallel q' \\
\Rightarrow & (* \sigma = \epsilon *) \\
& \exists p', q' : p \parallel q \xrightarrow{\sigma} p' \parallel q' \\
\Rightarrow & (* \text{Given } r = p' \parallel q' *) \\
& p \parallel q \xrightarrow{\sigma} r
\end{aligned}$$

Induction step: We assume the proposition holds for σ' in $\sigma = a \cdot \sigma', a \in L_\delta$.
We identify three cases:

1. Assume $a \in L_p^\delta \cap L_q^\delta (= (L_p \cap L_q) \cup \{\delta\})$:

$$\begin{aligned}
& \exists p', q' : p \xrightarrow{(a \cdot \sigma') \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{(a \cdot \sigma') \upharpoonright L_q^\delta} q' \\
\Rightarrow & (* \text{Definition A.2 of projection } *) \\
& \exists p_2, p', q_2, q' : p \xrightarrow{a} p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{a} q_2 \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \\
\Rightarrow & (* \text{Definition of } \xrightarrow{a} *) \\
& \exists p_1, p_2, p', q_1, q_2, q' : \\
& p \xrightarrow{\epsilon} p_1 \xrightarrow{a} p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\epsilon} q_1 \xrightarrow{a} q_2 \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \\
\Rightarrow & (* \text{Basic step } *) \\
& \exists p_1, p_2, p', q_1, q_2, q' : \\
& p \parallel q \xrightarrow{\epsilon} p_1 \parallel q_1 \wedge p_1 \xrightarrow{a} p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q_1 \xrightarrow{a} q_2 \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \\
\Rightarrow & (* \text{Proposition A.1.4 } *) \\
& \exists p_1, p_2, p', q_1, q_2, q' : \\
& p \parallel q \xrightarrow{\epsilon} p_1 \parallel q_1 \xrightarrow{a} p_2 \parallel q_2 \wedge p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q_2 \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \\
\Rightarrow & (* \text{Definition of } \xrightarrow{\epsilon} *) \\
& \exists p_2, p', q_2, q' : p \parallel q \xrightarrow{a} p_2 \parallel q_2 \wedge p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q_2 \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \\
\Rightarrow & (* \text{Induction } *) \\
& \exists p_2, p', q_2, q' : p \parallel q \xrightarrow{a} p_2 \parallel q_2 \wedge p_2 \parallel q_2 \xrightarrow{\sigma'} p' \parallel q' \\
\Rightarrow & (* \text{Definition of } \xrightarrow{a} *) \\
& \exists p', q' : p \parallel q \xrightarrow{a \cdot \sigma'} p' \parallel q' \\
\Rightarrow & (* \sigma = a \cdot \sigma' *) \\
& \exists p', q' : p \parallel q \xrightarrow{\sigma} p' \parallel q' \\
\Rightarrow & (* \text{Given } r = p' \parallel q' *) \\
& p \parallel q \xrightarrow{\sigma} r
\end{aligned}$$

2. Assume $a \in L_p^\delta \setminus L_q^\delta (= L_p \setminus L_q)$:
- $$\begin{aligned} & \exists p', q' : p \xrightarrow{(a \cdot \sigma') \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{(a \cdot \sigma') \upharpoonright L_q^\delta} q' \\ \Rightarrow & (*\text{Definition A.2 of projection } *) \\ & \exists p_2, p', q' : p \xrightarrow{a} p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \\ \Rightarrow & (*\text{Definition of } \Rightarrow *) \\ & \exists p_1, p_2, p', q' : p \xrightarrow{\epsilon} p_1 \xrightarrow{a} p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \\ \Rightarrow & (*\text{Basic step } *) \\ & \exists p_1, p_2, p', q' : p \parallel q \xrightarrow{\epsilon} p_1 \parallel q \wedge p_1 \xrightarrow{a} p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \\ \Rightarrow & (*\text{Proposition A.1.1 } *) \\ & \exists p_1, p_2, p', q' : p \parallel q \xrightarrow{\epsilon} p_1 \parallel q \xrightarrow{a} p_2 \parallel q \wedge p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \\ \Rightarrow & (*\text{Definition of } \Rightarrow *) \\ & \exists p_2, p', q' : p \parallel q \xrightarrow{a} p_2 \parallel q \wedge p_2 \xrightarrow{\sigma' \upharpoonright L_p^\delta} p' \wedge q \xrightarrow{\sigma' \upharpoonright L_q^\delta} q' \\ \Rightarrow & (*\text{Induction } *) \\ & \exists p_2, p', q' : p \parallel q \xrightarrow{a} p_2 \parallel q \wedge p_2 \parallel q \xrightarrow{\sigma'} p' \parallel q' \\ \Rightarrow & (*\text{Definition of } \Rightarrow *) \\ & \exists p', q' : p \parallel q \xrightarrow{a \cdot \sigma'} p' \parallel q' \\ \Rightarrow & (*\sigma = a \cdot \sigma' *) \\ & \exists p', q' : p \parallel q \xrightarrow{\sigma} p' \parallel q' \\ \Rightarrow & (*\text{Given } r = p' \parallel q' *) \\ & p \parallel q \xrightarrow{\sigma} r \end{aligned}$$
3. Assume $a \in L_q^\delta \setminus L_p^\delta$. This is symmetric with the previous case.

Lemma A.4. *Let $i, s \in \mathcal{IOTS}(I, U)$, then:*

$$i \text{ ioco } s \Leftrightarrow \text{Straces}(i) \subseteq \text{Straces}(s)$$

Proof.

only if: Proof by induction on the structure of σ , let $\sigma \in \text{Straces}(i)$.

Basic Step: $\sigma = \epsilon$.

$\epsilon \in \text{Straces}(s)$ trivially holds.

Induction step : We identify two cases:

1. $\sigma = \rho \cdot a$ with $a \in I$:

By induction $\rho \in \text{Straces}(s)$, so $\exists s' : s \xrightarrow{\rho} s'$, and since $s \in \mathcal{IOTS}(I, U)$, $s' \xrightarrow{a}$ always holds. Hence, $\rho \cdot a \in \text{Straces}(s)$.

2. $\sigma = \rho \cdot x$ with $x \in U \cup \{\delta\}$:

If $i \xrightarrow{\rho \cdot x}$ then by definition 5 $x \in \text{out}(i \text{ after } \rho)$, and since $i \text{ ioco } s$ and by induction $\rho \in \text{Straces}(s)$ we can conclude that $x \in \text{out}(s \text{ after } \rho)$.

Hence, $s \xrightarrow{\rho \cdot x}$, and $\rho \cdot x \in \text{Straces}(s)$.

if: Let $\sigma \in \text{Straces}(s)$ and $x \in \text{out}(i \text{ after } \sigma)$, then $i \xrightarrow{\sigma \cdot x}$, which implies $\sigma \cdot x \in \text{Straces}(i)$, hence $\sigma \cdot x \in \text{Straces}(s)$ and $s \xrightarrow{\sigma \cdot x}$, from which it follows that $x \in \text{out}(s \text{ after } \sigma)$.

Theorem 9 Let $s_1, i_1 \in \mathcal{IOTS}(I_1, U_1), s_2, i_2 \in \mathcal{IOTS}(I_2, U_2)$ with $I_1 \cap I_2 = U_1 \cap U_2 = \emptyset$.

$$i_1 \mathbf{ioco} s_1 \wedge i_2 \mathbf{ioco} s_2 \Rightarrow i_1 \parallel i_2 \mathbf{ioco} s_1 \parallel s_2$$

Proof. To be proved according to lemma A.4:

$$\begin{aligned} \text{Straces}(i_1) \subseteq \text{Straces}(s_1) \wedge \text{Straces}(i_2) \subseteq \text{Straces}(s_2) \\ \Rightarrow \text{Straces}(i_1 \parallel i_2) \subseteq \text{Straces}(s_1 \parallel s_2) \end{aligned}$$

$$\begin{aligned} & \sigma \in \text{Straces}(i_1 \parallel i_2) \\ \Rightarrow & (*\text{Definition of Straces } *) \\ & i_1 \parallel i_2 \xrightarrow{\sigma} \\ \Rightarrow & (*\text{Proposition A.3 } *) \\ & i_1 \xrightarrow{\sigma \upharpoonright L_{i_1}^\delta} \wedge i_2 \xrightarrow{\sigma \upharpoonright L_{i_2}^\delta} \\ \Rightarrow & (*\text{premise } *) \\ & s_1 \xrightarrow{\sigma \upharpoonright L_{s_1}^\delta} \wedge s_2 \xrightarrow{\sigma \upharpoonright L_{s_2}^\delta} \\ \Rightarrow & (*\text{Proposition A.3 } *) \\ & s_1 \parallel s_2 \xrightarrow{\sigma} \\ \Rightarrow & (*\text{Definition of Straces } *) \\ & \sigma \in \text{Straces}(s_1 \parallel s_2) \end{aligned}$$

A.2 Proofs of Section 4.2 (Congruence properties for hiding)

Definition A.5. The hiding of $\sigma \in L_\delta^*$ with $A \subseteq L$, denoted as $\sigma \setminus A$, is defined as follows:

$$\sigma \setminus A =_{\text{def}} \begin{cases} \epsilon & \sigma = \epsilon \\ \sigma' \setminus A & \sigma = a \cdot \sigma' \text{ with } a \in A \\ a \cdot (\sigma' \setminus A) & \sigma = a \cdot \sigma' \text{ with } a \notin A \\ \delta \cdot (\sigma' \setminus A) & \sigma = \delta \cdot \sigma' \end{cases}$$

Proposition A.6. Let $p \in \mathcal{LTS}(I, U)$ where U is partitioned in sets U_1 and U_2 .

1. $p \xrightarrow{a} p'$ with $a \in I \cup U_1 \Rightarrow \mathbf{hide} U_2 \text{ in } p \xrightarrow{a} \mathbf{hide} U_2 \text{ in } p'$
2. $p \xrightarrow{a} p'$ with $a \in U_2 \Rightarrow \mathbf{hide} U_2 \text{ in } p \xrightarrow{a} \mathbf{hide} U_2 \text{ in } p'$
3. $p \xrightarrow{\tau} p' \Rightarrow \mathbf{hide} U_2 \text{ in } p \xrightarrow{\tau} \mathbf{hide} U_2 \text{ in } p'$
4. $p \xrightarrow{\delta} p' \Rightarrow \mathbf{hide} U_2 \text{ in } p \xrightarrow{\delta} \mathbf{hide} U_2 \text{ in } p'$
5. $\mathbf{hide} U_2 \text{ in } p \xrightarrow{a} q$ with $a \in I \cup U_1 \Rightarrow \exists p' : p \xrightarrow{a} p' \wedge q = \mathbf{hide} U_2 \text{ in } p'$
6. $\mathbf{hide} U_2 \text{ in } p \xrightarrow{\tau} q \Rightarrow (\exists p' : p \xrightarrow{\tau} p' \wedge q = \mathbf{hide} U_2 \text{ in } p') \vee (\exists p', a \in U_2 : p \xrightarrow{a} p' \wedge q = \mathbf{hide} U_2 \text{ in } p')$
7. $\mathbf{hide} U_2 \text{ in } p \xrightarrow{\delta} q \Rightarrow p \xrightarrow{\delta} p \wedge q = \mathbf{hide} U_2 \text{ in } p$

Proof.

1. $p \xrightarrow{a} p' \wedge a \in I \cup U_1$
 \Rightarrow (*Definition of hide *)
 $\mathbf{hide} U_2 \text{ in } p \xrightarrow{a} \mathbf{hide} U_2 \text{ in } p'$
2. $p \xrightarrow{a} p' \wedge a \in U_2$
 \Rightarrow (*Definition of hide *)
 $\mathbf{hide} U_2 \text{ in } p \xrightarrow{\tau} \mathbf{hide} U_2 \text{ in } p'$
3. $p \xrightarrow{\tau} p'$
 \Rightarrow (*Definition of hide *)
 $\mathbf{hide} U_2 \text{ in } p \xrightarrow{\tau} \mathbf{hide} U_2 \text{ in } p'$
4. $p \xrightarrow{\delta} p'$
 \Rightarrow (*Definition of quiescence *)
 $(\forall \mu \in U \cup \{\tau\} : p \xrightarrow{\mu})$
 \Rightarrow (*Definition of hide *)
 $(\forall \mu \in U_1 \cup \{\tau\} : \mathbf{hide} U_2 \text{ in } p \xrightarrow{\mu})$
 \Rightarrow (*Definition of quiescence *)
 $\mathbf{hide} U_2 \text{ in } p \xrightarrow{\delta} \mathbf{hide} U_2 \text{ in } p$
5. $\mathbf{hide} U_2 \text{ in } p \xrightarrow{a} q$
 \Rightarrow (*Definition of hide *)
 $\exists p' : p \xrightarrow{a} p' \wedge q = \mathbf{hide} U_2 \text{ in } p'$
6. $\mathbf{hide} U_2 \text{ in } p \xrightarrow{\tau} q$
 \Rightarrow (*Definition of hide *)
 $(\exists p' : p \xrightarrow{\tau} p' \wedge q = \mathbf{hide} U_2 \text{ in } p') \vee$
 $(\exists p', \exists a \in U_2 : p \xrightarrow{a} p' \wedge q = \mathbf{hide} U_2 \text{ in } p')$
7. $\mathbf{hide} U_2 \text{ in } p \xrightarrow{\delta} q$
 \Rightarrow (*Definition of quiescence *)
 $\forall \mu \in U_1 \cup \{\tau\} : \mathbf{hide} U_2 \text{ in } p \xrightarrow{\mu} \wedge q = \mathbf{hide} U_2 \text{ in } p$
 \Rightarrow (*Definition of hide *)
 $(\forall \mu \in U_1 : p \xrightarrow{\mu}) \wedge (\forall \mu \in U_2 : p \xrightarrow{\mu}) \wedge$
 $p \xrightarrow{\tau} \wedge q = \mathbf{hide} U_2 \text{ in } p$
 \Rightarrow (*Logical axioms *)
 $\forall \mu \in U \cup \{\tau\} : p \xrightarrow{\mu} \wedge q = \mathbf{hide} U_2 \text{ in } p$
 \Rightarrow (*Definition of quiescence *)
 $p \xrightarrow{\delta} p \wedge q = \mathbf{hide} U_2 \text{ in } p$

Proposition A.7. *Let $p \in \mathcal{LTS}(I, U)$ where U is partitioned into U_1 and U_2 ; let $\sigma \in L_\delta^*$ be arbitrary.*

1. $p \xrightarrow{\sigma} p' \Rightarrow \mathbf{hide} U_2 \text{ in } p \xrightarrow{\sigma \setminus U_2} \mathbf{hide} U_2 \text{ in } p'$
2. $\mathbf{hide} U_2 \text{ in } p \xrightarrow{\sigma} q \Rightarrow \exists p', \exists \sigma' \in L_\delta^* : p \xrightarrow{\sigma'} p' \wedge q = \mathbf{hide} U_2 \text{ in } p' \wedge \sigma = \sigma' \setminus U_2$

Proof.

1. By induction on the structure of σ :

$\sigma = \epsilon$:

Using $\epsilon \setminus U_2 = \epsilon$, the proposition reduces to:

$$p \xRightarrow{\epsilon} p' \Rightarrow \mathbf{hide} U_2 \mathbf{in} p \xRightarrow{\epsilon} \mathbf{hide} U_2 \mathbf{in} p' \quad (2)$$

which, using the definition of $\xRightarrow{\epsilon}$, is rewritten to:

$$p \xrightarrow{\tau^n} p' \Rightarrow \mathbf{hide} U_2 \mathbf{in} p \xRightarrow{\epsilon} \mathbf{hide} U_2 \mathbf{in} p' \quad (3)$$

which is proved by induction on n :

$n = 0$:

$$p \xrightarrow{\tau^n} p'$$

$$\Rightarrow (*n = 0 \text{ and from definition of } \xRightarrow{\epsilon} : p \xrightarrow{\tau^0} p' \Leftrightarrow p = p' *)$$

$$p = p'$$

$$\Rightarrow (*\text{definition of } \xRightarrow{\epsilon} *)$$

$$\mathbf{hide} U_2 \mathbf{in} p \xRightarrow{\epsilon} \mathbf{hide} U_2 \mathbf{in} p'$$

$n = n' + 1$:

$$p \xrightarrow{\tau^n} p'$$

$$\Rightarrow (*n = n' + 1 \text{ and definition of } \xRightarrow{\epsilon} *)$$

$$\exists p_1 : p \xrightarrow{\tau} p_1 \wedge p_1 \xrightarrow{\tau^{n'}} p'$$

$$\Rightarrow (*\text{proposition A.6.3} *)$$

$$\mathbf{hide} U_2 \mathbf{in} p \xrightarrow{\tau} \mathbf{hide} U_2 \mathbf{in} p_1 \wedge p_1 \xrightarrow{\tau^{n'}} p'$$

$$\Rightarrow (*\text{induction on equation (3)} *)$$

$$\mathbf{hide} U_2 \mathbf{in} p \xrightarrow{\tau} \mathbf{hide} U_2 \mathbf{in} p_1 \wedge \mathbf{hide} U_2 \mathbf{in} p_1 \xRightarrow{\epsilon} \mathbf{hide} U_2 \mathbf{in} p'$$

$$\Rightarrow (*\text{definition of } \xRightarrow{\epsilon} *)$$

$$\mathbf{hide} U_2 \mathbf{in} p \xRightarrow{\epsilon} \mathbf{hide} U_2 \mathbf{in} p'$$

$\sigma = a \cdot \rho$, with $a \in I \cup U_1$, $\rho \in L_\delta^*$:

$$\begin{aligned}
& p \xrightarrow{\sigma} p' \\
\Rightarrow & (* \sigma = a \cdot \rho *) \\
& p \xrightarrow{a \cdot \rho} p' \\
\Rightarrow & (* \text{definition of } \xrightarrow{a \cdot \rho} *) \\
& \exists p_1, p_2 : p \xrightarrow{\epsilon} p_1 \wedge p_1 \xrightarrow{a} p_2 \wedge p_2 \xrightarrow{\rho} p' \\
\Rightarrow & (* \text{equation (2)} *) \\
& \exists p_1, p_2 : \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{\epsilon} \mathbf{hide } U_2 \mathbf{ in } p_1 \wedge p_1 \xrightarrow{a} p_2 \wedge p_2 \xrightarrow{\rho} p' \\
\Rightarrow & (* \text{proposition A.6.1} *) \\
& \exists p_1, p_2 : \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{\epsilon} \mathbf{hide } U_2 \mathbf{ in } p_1 \wedge \\
& \mathbf{hide } U_2 \mathbf{ in } p_1 \xrightarrow{a} \mathbf{hide } U_2 \mathbf{ in } p_2 \wedge p_2 \xrightarrow{\rho} p' \\
\Rightarrow & (* \text{induction} *) \\
& \exists p_1, p_2 : \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{\epsilon} \mathbf{hide } U_2 \mathbf{ in } p_1 \wedge \\
& \mathbf{hide } U_2 \mathbf{ in } p_1 \xrightarrow{a} \mathbf{hide } U_2 \mathbf{ in } p_2 \wedge \\
& \mathbf{hide } U_2 \mathbf{ in } p_2 \xrightarrow{\rho \setminus U_2} \mathbf{hide } U_2 \mathbf{ in } p' \\
\Rightarrow & (* \text{definition of } \xrightarrow{a \cdot (\rho \setminus U_2)} *) \\
& \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{a \cdot (\rho \setminus U_2)} \mathbf{hide } U_2 \mathbf{ in } p' \\
\Rightarrow & (* \text{definition A.5} *) \\
& \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{(a \cdot \rho) \setminus U_2} \mathbf{hide } U_2 \mathbf{ in } p' \\
\Rightarrow & (* \sigma = a \cdot \rho *) \\
& \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{\sigma \setminus U_2} \mathbf{hide } U_2 \mathbf{ in } p'
\end{aligned}$$

$\sigma = a \cdot \rho$, with $a \in U_2, \rho \in L_\delta^*$:

$$\begin{aligned}
& p \xrightarrow{\sigma} p' \\
\Rightarrow & (* \sigma = a \cdot \rho *) \\
& p \xrightarrow{a \cdot \rho} p' \\
\Rightarrow & (* \text{definition } \xrightarrow{a \cdot \rho} *) \\
& \exists p_1, p_2 : p \xrightarrow{\epsilon} p_1 \wedge p_1 \xrightarrow{a} p_2 \wedge p_2 \xrightarrow{\rho} p' \\
\Rightarrow & (* \text{equation (2) } *) \\
& \exists p_1, p_2 : \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{\epsilon} \mathbf{hide } U_2 \mathbf{ in } p_1 \wedge p_1 \xrightarrow{a} p_2 \wedge p_2 \xrightarrow{\rho} p' \\
\Rightarrow & (* \text{proposition A.6.2 } *) \\
& \exists p_1, p_2 : \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{\epsilon} \mathbf{hide } U_2 \mathbf{ in } p_1 \wedge \\
& \mathbf{hide } U_2 \mathbf{ in } p_1 \xrightarrow{\tau} \mathbf{hide } U_2 \mathbf{ in } p_2 \wedge p_2 \xrightarrow{\rho} p' \\
\Rightarrow & (* \text{induction } *) \\
& \exists p_1, p_2 : \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{\epsilon} \mathbf{hide } U_2 \mathbf{ in } p_1 \wedge \\
& \mathbf{hide } U_2 \mathbf{ in } p_1 \xrightarrow{\tau} \mathbf{hide } U_2 \mathbf{ in } p_2 \wedge \\
& \mathbf{hide } U_2 \mathbf{ in } p_2 \xrightarrow{\rho \setminus U_2} \mathbf{hide } U_2 \mathbf{ in } p' \\
\Rightarrow & (* \text{definition of } \xrightarrow{\rho \setminus U_2} *) \\
& \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{\rho \setminus U_2} \mathbf{hide } U_2 \mathbf{ in } p' \\
\Rightarrow & (* \text{definition A.5 } *) \\
& \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{(a \cdot \rho) \setminus U_2} \mathbf{hide } U_2 \mathbf{ in } p' \\
\Rightarrow & (* \sigma = a \cdot \rho *) \\
& \mathbf{hide } U_2 \mathbf{ in } p \xrightarrow{\sigma \setminus U_2} \mathbf{hide } U_2 \mathbf{ in } p'
\end{aligned}$$

$$\begin{aligned}
& \sigma = \delta \cdot \rho, \text{ with } \rho \in L_\delta^*: \\
& p \xrightarrow{\sigma} p' \\
& \Rightarrow (* \sigma = \delta \cdot \rho *) \\
& p \xrightarrow{\delta \cdot \rho} p' \\
& \Rightarrow (* \text{definition } \xrightarrow{\delta \cdot \rho} \text{ and definition of quiescence } *) \\
& \exists p_1 : p \xrightarrow{\epsilon} p_1 \wedge p_1 \xrightarrow{\delta} p_1 \wedge p_1 \xrightarrow{\rho} p' \\
& \Rightarrow (* \text{equation (2)} *) \\
& \exists p_1 : \mathbf{hide } U_2 \text{ in } p \xrightarrow{\epsilon} \mathbf{hide } U_2 \text{ in } p_1 \wedge p_1 \xrightarrow{\delta} p_1 \wedge p_1 \xrightarrow{\rho} p' \\
& \Rightarrow (* \text{proposition A.6.4} *) \\
& \exists p_1 : \mathbf{hide } U_2 \text{ in } p \xrightarrow{\epsilon} \mathbf{hide } U_2 \text{ in } p_1 \wedge \\
& \mathbf{hide } U_2 \text{ in } p_1 \xrightarrow{\delta} \mathbf{hide } U_2 \text{ in } p_1 \wedge p_1 \xrightarrow{\rho} p' \\
& \Rightarrow (* \text{induction } *) \\
& \exists p_1 : \mathbf{hide } U_2 \text{ in } p \xrightarrow{\epsilon} \mathbf{hide } U_2 \text{ in } p_1 \wedge \\
& \mathbf{hide } U_2 \text{ in } p_1 \xrightarrow{\delta} \mathbf{hide } U_2 \text{ in } p_1 \wedge \\
& \mathbf{hide } U_2 \text{ in } p_1 \xrightarrow{\rho \setminus U_2} \mathbf{hide } U_2 \text{ in } p' \\
& \Rightarrow (* \text{definition of } \xrightarrow{\delta \cdot (\rho \setminus U_2)} *) \\
& \mathbf{hide } U_2 \text{ in } p \xrightarrow{\delta \cdot (\rho \setminus U_2)} \mathbf{hide } U_2 \text{ in } p' \\
& \Rightarrow (* \text{definition A.5} *) \\
& \mathbf{hide } U_2 \text{ in } p \xrightarrow{(\delta \cdot \rho) \setminus U_2} \mathbf{hide } U_2 \text{ in } p' \\
& \Rightarrow (* \sigma = \delta \cdot \rho *) \\
& \mathbf{hide } U_2 \text{ in } p \xrightarrow{\sigma \setminus U_2} \mathbf{hide } U_2 \text{ in } p'
\end{aligned}$$

2. By induction on the structure of σ (Note that $\sigma \in (I \cup U_1 \cup \{\delta\})^*$):

$\sigma = \epsilon$:

The proposition reduces to:

$$\begin{aligned}
\mathbf{hide } U_2 \text{ in } p \xrightarrow{\epsilon} q & \Rightarrow \exists p', \exists \sigma' \in L_\delta^* : \\
p & \xrightarrow{\sigma'} p' \wedge q = \mathbf{hide } U_2 \text{ in } p' \wedge \epsilon = \sigma' \setminus U_2
\end{aligned} \tag{4}$$

which, using the definition of $\xrightarrow{\epsilon}$, is rewritten to:

$$\begin{aligned}
\mathbf{hide } U_2 \text{ in } p \xrightarrow{\tau^n} q & \Rightarrow \exists p', \exists \sigma' \in L_\delta^* : \\
p & \xrightarrow{\sigma'} p' \wedge q = \mathbf{hide } U_2 \text{ in } p' \wedge \epsilon = \sigma' \setminus U_2
\end{aligned} \tag{5}$$

which is proved by induction on n :

$n = 0$:

$$\begin{aligned}
& \mathbf{hide } U_2 \text{ in } p \xrightarrow{\tau^0} q \\
& \Rightarrow (* n = 0 \text{ and from definition of } \xrightarrow{\tau^0} : p \xrightarrow{\tau^0} p' \Leftrightarrow p = p' *) \\
& \exists p' = p, \exists \sigma' = \epsilon : p \xrightarrow{\sigma'} p' \wedge q = \mathbf{hide } U_2 \text{ in } p' \wedge \epsilon = \sigma' \setminus U_2
\end{aligned}$$

$n = n' + 1$:

$$\begin{aligned}
& \mathbf{hide} U_2 \mathbf{in} p \xrightarrow{\tau^n} q \\
\Rightarrow & (*n = n' + 1 \text{ and definition of } \xrightarrow{\tau^n} *) \\
& \exists q_1 : \mathbf{hide} U_2 \mathbf{in} p \xrightarrow{\tau} q_1 \wedge q_1 \xrightarrow{\tau^{n'}} q \\
\Rightarrow & (*\text{proposition A.6.6} *) \\
& \exists q_1 : (\exists p_1 : (p \xrightarrow{\tau} p_1 \vee \exists a \in U_2 : p \xrightarrow{a} p_1) \wedge \\
& \quad q_1 = \mathbf{hide} U_2 \mathbf{in} p_1) \wedge q_1 \xrightarrow{\tau^{n'}} q \\
\Rightarrow & (*\text{substitution} *) \\
& \exists p_1 : (p \xrightarrow{\tau} p_1 \vee \exists a \in U_2 : p \xrightarrow{a} p_1) \wedge \mathbf{hide} U_2 \mathbf{in} p_1 \xrightarrow{\tau^{n'}} q \\
\Rightarrow & (*\text{induction on equation (5)} *) \\
& \exists p_1 : (p \xrightarrow{\tau} p_1 \vee \exists a \in U_2 : p \xrightarrow{a} p_1) \wedge \\
& \quad (\exists p_2, \exists \sigma_1 \in L_\delta^* : \\
& \quad p_1 \xrightarrow{\sigma_1} p_2 \wedge q = \mathbf{hide} U_2 \mathbf{in} p_2 \wedge \epsilon = \sigma_1 \setminus U_2) \\
\Rightarrow & (*\text{logical manipulation} *) \\
& (\exists p_1, p_2, \exists \sigma_1 \in L_\delta^* : \\
& \quad p \xrightarrow{\tau} p_1 \wedge p_1 \xrightarrow{\sigma_1} p_2 \wedge q = \mathbf{hide} U_2 \mathbf{in} p_2 \wedge \epsilon = \sigma_1 \setminus U_2) \\
& \vee (\exists p_1, p_2, \exists \sigma_1 \in L_\delta^*, \exists a \in U_2 : \\
& \quad p \xrightarrow{a} p_1 \wedge p_1 \xrightarrow{\sigma_1} p_2 \wedge q = \mathbf{hide} U_2 \mathbf{in} p_2 \wedge \epsilon = \sigma_1 \setminus U_2) \\
\Rightarrow & (*\text{Definition of } \Rightarrow *) \\
& (\exists p' = p_2, \sigma' = \sigma_1 : p \xrightarrow{\sigma'} p' \wedge q = \mathbf{hide} U_2 \mathbf{in} p' \wedge \epsilon = \sigma' \setminus U_2) \\
& \vee (\exists p' = p_2, \sigma' = a \cdot \sigma_1 : p \xrightarrow{\sigma'} p' \wedge q = \mathbf{hide} U_2 \mathbf{in} p' \wedge \epsilon = \sigma' \setminus U_2)
\end{aligned}$$

$$\begin{aligned}
& \sigma = a \cdot \rho, \text{ with } a \in I \cup U_1: \\
& \quad \mathbf{hide } U_2 \text{ in } p \xrightarrow{\sigma} q \\
& \Rightarrow (*\sigma = a \cdot \rho *) \\
& \quad \mathbf{hide } U_2 \text{ in } p \xrightarrow{a \cdot \rho} q \\
& \Rightarrow (*\text{definition of } \xrightarrow{a \cdot \rho} *) \\
& \quad \exists q_1, q_2 : \mathbf{hide } U_2 \text{ in } p \xrightarrow{\epsilon} q_1 \wedge q_1 \xrightarrow{a} q_2 \wedge q_2 \xrightarrow{\rho} q \\
& \Rightarrow (*\text{equation (4)} *) \\
& \quad \exists q_1, q_2 : (\exists p_1, \exists \sigma_1 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge q_1 = \mathbf{hide } U_2 \text{ in } p_1 \wedge \epsilon = \sigma_1 \setminus U_2) \wedge \\
& \quad q_1 \xrightarrow{a} q_2 \wedge q_2 \xrightarrow{\rho} q \\
& \Rightarrow (*\text{substitution } *) \\
& \quad \exists q_2, p_1, \exists \sigma_1 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge \epsilon = \sigma_1 \setminus U_2 \wedge \mathbf{hide } U_2 \text{ in } p_1 \xrightarrow{a} q_2 \wedge q_2 \xrightarrow{\rho} q \\
& \Rightarrow (*\text{proposition A.6.5 } *) \\
& \quad \exists q_2, p_1, \exists \sigma_1 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge \epsilon = \sigma_1 \setminus U_2 \wedge \\
& \quad (\exists p_2 : p_1 \xrightarrow{a} p_2 \wedge q_2 = \mathbf{hide } U_2 \text{ in } p_2) \wedge q_2 \xrightarrow{\rho} q \\
& \Rightarrow (*\text{substitution } *) \\
& \quad \exists p_1, p_2, \exists \sigma_1 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge \epsilon = \sigma_1 \setminus U_2 \wedge p_1 \xrightarrow{a} p_2 \wedge \mathbf{hide } U_2 \text{ in } p_2 \xrightarrow{\rho} q \\
& \Rightarrow (*\text{induction } *) \\
& \quad \exists p_1, p_2, \exists \sigma_1 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge \epsilon = \sigma_1 \setminus U_2 \wedge p_1 \xrightarrow{a} p_2 \wedge \\
& \quad (\exists p_3, \exists \sigma_3 \in L_\delta^* : \\
& \quad p_2 \xrightarrow{\sigma_3} p_3 \wedge q = \mathbf{hide } U_2 \text{ in } p_3 \wedge \rho = \sigma_3 \setminus U_2) \\
& \Rightarrow (*\text{logical manipulation } *) \\
& \quad \exists p_1, p_2, p_3, \exists \sigma_1, \sigma_3 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge p_1 \xrightarrow{a} p_2 \wedge p_2 \xrightarrow{\sigma_3} p_3 \wedge \\
& \quad q = \mathbf{hide } U_2 \text{ in } p_3 \wedge \epsilon = \sigma_1 \setminus U_2 \wedge \rho = \sigma_3 \setminus U_2 \\
& \Rightarrow (*(\sigma_1 \cdot a \cdot \sigma_3) \setminus U_2 = (\sigma_1 \setminus U_2) \cdot (a \setminus U_2) \cdot (\sigma_3 \setminus U_2) = \epsilon \cdot a \cdot \rho = \sigma *) \\
& \quad \exists p' = p_3, \sigma' = \sigma_1 \cdot a \cdot \sigma_3 : \\
& \quad p \xrightarrow{\sigma'} p' \wedge q = \mathbf{hide } U_2 \text{ in } p' \wedge \sigma = \sigma' \setminus U_2
\end{aligned}$$

$$\begin{aligned}
& \sigma = \delta \cdot \rho: \\
& \quad \mathbf{hide} U_2 \mathbf{in} p \xrightarrow{\sigma} q \\
& \Rightarrow (*\sigma = \delta \cdot \rho *) \\
& \quad \mathbf{hide} U_2 \mathbf{in} p \xrightarrow{\delta \cdot \rho} q \\
& \Rightarrow (*\text{definition of } \xrightarrow{\delta \cdot \rho} *) \\
& \quad \exists q_1, q_2 : \mathbf{hide} U_2 \mathbf{in} p \xrightarrow{\epsilon} q_1 \wedge q_1 \xrightarrow{\delta} q_2 \wedge q_2 \xrightarrow{\rho} q \\
& \Rightarrow (*\text{equation (4)} *) \\
& \quad \exists q_1, q_2 : (\exists p_1, \exists \sigma_1 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge q_1 = \mathbf{hide} U_2 \mathbf{in} p_1 \wedge \epsilon = \sigma_1 \setminus U_2) \wedge \\
& \quad q_1 \xrightarrow{\delta} q_2 \wedge q_2 \xrightarrow{\rho} q \\
& \Rightarrow (*\text{substitution} *) \\
& \quad \exists q_2, p_1, \exists \sigma_1 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge \epsilon = \sigma_1 \setminus U_2 \wedge \mathbf{hide} U_2 \mathbf{in} p_1 \xrightarrow{\delta} q_2 \wedge q_2 \xrightarrow{\rho} q \\
& \Rightarrow (*\text{proposition A.6.7} *) \\
& \quad \exists q_2, p_1, \exists \sigma_1 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge \epsilon = \sigma_1 \setminus U_2 \wedge p_1 \xrightarrow{\delta} p_1 \wedge \\
& \quad q_2 = \mathbf{hide} U_2 \mathbf{in} p_1 \wedge q_2 \xrightarrow{\rho} q \\
& \Rightarrow (*\text{substitution} *) \\
& \quad \exists p_1, \exists \sigma_1 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge \epsilon = \sigma_1 \setminus U_2 \wedge p_1 \xrightarrow{\delta} p_1 \wedge \mathbf{hide} U_2 \mathbf{in} p_1 \xrightarrow{\rho} q \\
& \Rightarrow (*\text{induction} *) \\
& \quad \exists p_1, \exists \sigma_1 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge \epsilon = \sigma_1 \setminus U_2 \wedge p_1 \xrightarrow{\delta} p_1 \wedge \\
& \quad (\exists p_3, \exists \sigma_3 \in L_\delta^* : \\
& \quad p_1 \xrightarrow{\sigma_3} p_3 \wedge q = \mathbf{hide} U_2 \mathbf{in} p_3 \wedge \rho = \sigma_3 \setminus U_2) \\
& \Rightarrow (*\text{logical manipulation} *) \\
& \quad \exists p_1, p_3, \exists \sigma_1, \sigma_3 \in L_\delta^* : \\
& \quad p \xrightarrow{\sigma_1} p_1 \wedge p_1 \xrightarrow{\delta} p_1 \wedge p_1 \xrightarrow{\sigma_3} p_3 \wedge \\
& \quad q = \mathbf{hide} U_2 \mathbf{in} p_3 \wedge \epsilon = \sigma_1 \setminus U_2 \wedge \rho = \sigma_3 \setminus U_2 \\
& \Rightarrow (*(\sigma_1 \cdot \delta \cdot \sigma_3) \setminus U_2 = (\sigma_1 \setminus U_2) \cdot (\delta \setminus U_2) \cdot (\sigma_3 \setminus U_2) = \epsilon \cdot \delta \cdot \rho = \sigma *) \\
& \quad \exists p' = p_3, \sigma' = \sigma_1 \cdot \delta \cdot \sigma_3 : \\
& \quad p \xrightarrow{\sigma'} p' \wedge q = \mathbf{hide} U_2 \mathbf{in} p' \wedge \sigma = \sigma' \setminus U_2
\end{aligned}$$

Theorem 11 If $i, s \in \mathcal{IOTS}(I, U)$ with $U_2 \subseteq U$, then:

$$i \text{ ioco } s \Rightarrow \mathbf{hide} U_2 \mathbf{in} i \text{ ioco } \mathbf{hide} U_2 \mathbf{in} s$$

Proof. To be proved according to lemma A.4:

$$\text{Straces}(i) \subseteq \text{Straces}(s) \Rightarrow \text{Straces}(\mathbf{hide} U_2 \mathbf{in} i) \subseteq \text{Straces}(\mathbf{hide} U_2 \mathbf{in} s)$$

which is straightforward (recall that $L = I \cup U$).

$$\begin{aligned}
& \sigma \in \text{Straces}(\mathbf{hide } U_2 \mathbf{ in } i) \\
\Rightarrow & (*\text{definition 5 } *) \\
& \mathbf{hide } U_2 \mathbf{ in } i \xrightarrow{\sigma} \\
\Rightarrow & (*\text{proposition A.7.2 } *) \\
& \exists \sigma' \in L_\delta^* : i \xrightarrow{\sigma'} \wedge \sigma = \sigma' \setminus U_2 \\
\Rightarrow & (*\text{premise } *) \\
& \exists \sigma' \in L_\delta^* : s \xrightarrow{\sigma'} \wedge \sigma = \sigma' \setminus U_2 \\
\Rightarrow & (*\text{proposition A.7.1 } *) \\
& \exists \sigma' \in L_\delta^* : \mathbf{hide } U_2 \mathbf{ in } s \xrightarrow{\sigma' \setminus U_2} \wedge \sigma = \sigma' \setminus U_2 \\
\Rightarrow & (*\text{substitution } *) \\
& \mathbf{hide } U_2 \mathbf{ in } s \xrightarrow{\sigma} \\
\Rightarrow & (*\text{definition 5 } *) \\
& \sigma \in \text{Straces}(\mathbf{hide } U_2 \mathbf{ in } s)
\end{aligned}$$

A.3 Proofs of section 5

To improve the readability of the propositions and proofs in this chapter, we make a notational distinction between transitions of s and $\Xi(s)$. For transitions of the IOTS $\Xi(s)$, we use the subscript Ξ . For example, we use $q \xrightarrow{\lambda}_{\Xi} q'$ to denote $(q, \lambda, q') \in T_{\Xi(s)}$. Likewise we use \Rightarrow_{Ξ} for sequences of transitions. For transitions and sequences of transitions in s we use the standard notation without subscript.

Lemma A.8. $(U \cup \delta^* I)^* \delta^*$ is suffix-closed. Let $\sigma \in (U \cup \delta^* I)^* \delta^*$.

$$\forall \sigma_2 (\sigma = \sigma_1 \cdot \sigma_2) : \sigma_2 \in (U \cup \delta^* I)^* \delta^*$$

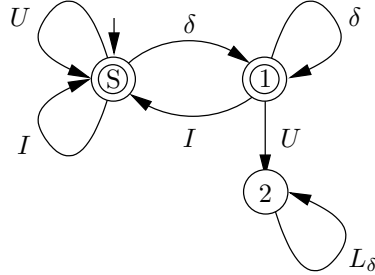


Fig. 9. Automaton that accepts the regular set $(U \cup \delta^* I)^* \delta^*$

Proof. In Figure 9 we show the automaton that accepts strings in the regular set $(U \cup \delta^* I)^* \delta^*$. The two accepting states are “S” (the starting state) and “1”. It is impossible to reach an accepting state from state “2”. So after σ_1 the

automaton is in state “S” or “1”. In “S”, being the start state, the entire regular set is accepted. In state “1” strings of the following form are accepted. Both are subsets of $(U \cup \delta^* I)^* \delta^*$.

1. $\delta^* \subseteq (U \cup \delta^* I)^* \delta^*$
2. $\delta^* I (U \cup \delta^* I)^* \delta^* \subseteq (U \cup \delta^* I)^* \delta^*$

Proposition 14 Let $s \in \mathcal{LTS}(I, U)$

$$\forall \sigma \in \text{Straces}(s) : \sigma \in (U \cup \delta^* I)^* \delta^*$$

Proof. The complement of $(U \cup \delta^* I)^* \delta^*$ with respect to L_δ^* are traces of the form $(U \cup \delta^* I)^* \delta^* (\delta U) L_\delta^*$. It is straightforward to verify that this is really the complement by interchanging the accepting and non accepting states in Figure 9. The result is an automaton that accepts the complement. So a trace in the complement is a trace that contains the sequence $\delta \cdot \lambda$ with $\lambda \in U$. Because of the semantics of quiescence this is of course an impossible trace for an LTS.

Proof by reductio ad absurdum. Suppose the proposition is not true. Let $\sigma = \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \in (U \cup \delta^* I)^* \delta^* (\delta U) L_\delta^*$ where $\sigma_1 \in (U \cup \delta^* I)^* \delta^*$, $\sigma_2 \in \delta U$, $\sigma_3 \in L_\delta^*$. Take an arbitrary s with $\sigma \in \text{Straces}(s)$, such that $s \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2 \cdot \sigma_3}$

$$\begin{aligned} & \sigma \in \text{Straces}(s) \wedge s \xrightarrow{\sigma_1} q_1 \xrightarrow{\delta} q_2 \\ \Rightarrow & (*\text{Definition } \delta \text{ } *) \\ & \sigma \in \text{Straces}(s) \wedge s \xrightarrow{\sigma_1} q_1 \wedge \forall \lambda \in U : q_1 \not\xrightarrow{\lambda} \wedge q_1 \not\xrightarrow{\tau} \wedge q_1 = q_2 \\ \Rightarrow & (*\text{Definition } \Rightarrow \text{ and } \sigma_2 \in \delta U \text{ } *) \\ & \sigma \in \text{Straces}(s) \wedge s \xrightarrow{\sigma_1} q_1 \wedge q_1 \xrightarrow{\sigma_2} \\ \Rightarrow & \text{Contradiction} \end{aligned}$$

Proposition 15

$$\forall \sigma \in L_\delta^*, q' \in Q_s : s \xrightarrow{\sigma} q' \Leftrightarrow \Xi(s) \xrightarrow{\sigma} \Xi q'$$

Proof.

Only if: Proof by induction on the structure of σ .

Basic step: $\sigma = \epsilon$. We prove the following stronger statement

$$\forall q, q' \in Q_s : q \xrightarrow{\tau^n} q' \Rightarrow q \xrightarrow{\tau^n} \Xi q' \tag{6}$$

which is proven by induction on n :

Basic step: $n = 0$

$$\begin{aligned} & q \xrightarrow{\tau^0} q' \\ \Rightarrow & (*q = q' \text{ and } q, q' \in Q_{\Xi(s)} \text{ } *) \\ & q \xrightarrow{\tau^0} \Xi q' \end{aligned}$$

Induction step: $n = n' + 1$. We assume that equation 6 holds for n' .

$$\begin{aligned}
& q \xrightarrow{\tau^{n'} \cdot \tau} q' \\
\Rightarrow & (* \text{Definition } \rightarrow *) \\
& \exists q_1 \in Q_s : q \xrightarrow{\tau^{n'}} q_1 \wedge q_1 \xrightarrow{\tau} q' \\
\Rightarrow & (* \text{Induction } *) \\
& \exists q_1 \in Q_s : q \xrightarrow{\tau^{n'}}_{\Xi} q_1 \wedge q_1 \xrightarrow{\tau} q' \\
\Rightarrow & (* \text{Definition of } \Xi \text{ (note that } (q_1, \tau, q') \in T_s \subseteq T_{\Xi(s)} \text{)} *) \\
& \exists q_1 \in Q_s : q \xrightarrow{\tau^{n'}}_{\Xi} q_1 \wedge q_1 \xrightarrow{\tau}_{\Xi} q' \\
\Rightarrow & (* \text{Definition } \rightarrow *) \\
& q \xrightarrow{\tau^{n'+1}}_{\Xi} q'
\end{aligned}$$

Induction step: $\sigma = \sigma' \cdot a$, where $a \in L_\delta$. We assume that the proposition holds for σ' . We identify two cases:

1. $a \in L$

$$\begin{aligned}
& s \xrightarrow{\sigma' \cdot a} q' \\
\Rightarrow & (* \text{Definition } \Rightarrow *) \\
& \exists q_1, q_2 \in Q_s : s \xrightarrow{\sigma'} q_1 \wedge q_1 \xrightarrow{a} q_2 \wedge q_2 \xrightarrow{\epsilon} q' \\
\Rightarrow & (* \text{Induction } *) \\
& \exists q_1, q_2 \in Q_s : \Xi(s) \xrightarrow{\sigma'}_{\Xi} q_1 \wedge q_1 \xrightarrow{a} q_2 \wedge q_2 \xrightarrow{\epsilon} q' \\
\Rightarrow & (* \text{Definition of } \Xi(s) \text{ (note that } (q_1, a, q_2) \in T_s \subseteq T_{\Xi(s)} \text{)} *) \\
& \exists q_1, q_2 \in Q_s : \Xi(s) \xrightarrow{\sigma'}_{\Xi} q_1 \wedge q_1 \xrightarrow{a}_{\Xi} q_2 \wedge q_2 \xrightarrow{\epsilon} q' \\
\Rightarrow & (* \text{Equation 6 } *) \\
& \exists q_1, q_2 \in Q_s : \Xi(s) \xrightarrow{\sigma'}_{\Xi} q_1 \wedge q_1 \xrightarrow{a}_{\Xi} q_2 \wedge q_2 \xrightarrow{\epsilon}_{\Xi} q' \\
\Rightarrow & (* \text{Definition } \Rightarrow *) \\
& \Xi(s) \xrightarrow{\sigma' \cdot a}_{\Xi} q'
\end{aligned}$$

2. $a = \delta$

$$\begin{aligned}
& s \xrightarrow{\sigma' \cdot \delta} q' \\
\Rightarrow & (* \text{Definition } \Rightarrow *) \\
& \exists q_1, q_2 \in Q_s : s \xrightarrow{\sigma'} q_1 \wedge q_1 \xrightarrow{\delta} q_2 \wedge q_2 \xrightarrow{\epsilon} q' \\
\Rightarrow & (* \text{Definition } \delta \text{ (} q_1 = q_2 = q' \text{)} *) \\
& s \xrightarrow{\sigma'} q' \wedge \forall \lambda \in U_\tau : q' \xrightarrow{\lambda} \\
\Rightarrow & (* \text{Induction, note that } q' \in Q_s *) \\
& \Xi(s) \xrightarrow{\sigma'}_{\Xi} q' \wedge \forall \lambda \in U_\tau : q' \xrightarrow{\lambda} \\
\Rightarrow & (* \text{Definition } \Xi, \text{ note that } \lambda \notin I \text{ and } q' \in Q_s *) \\
& \Xi(s) \xrightarrow{\sigma'}_{\Xi} q' \wedge \forall \lambda \in U_\tau : q' \xrightarrow{\lambda}_{\Xi} \\
\Rightarrow & (* \text{Definition } \delta *) \\
& \Xi(s) \xrightarrow{\sigma'}_{\Xi} q' \wedge q' \xrightarrow{\delta}_{\Xi} q' \\
\Rightarrow & (* \text{Definition } \Rightarrow *) \\
& \Xi(s) \xrightarrow{\sigma' \cdot \delta}_{\Xi} q'
\end{aligned}$$

if: Proof by induction on the structure of σ .

Basic step: $\sigma = \epsilon$ We prove the following stronger statement

$$\forall q \in Q_{\Xi(s)}, q' \in Q_s : q \xrightarrow{\tau^n}_{\Xi} q' \Rightarrow q \xrightarrow{\tau^n} q' \quad (7)$$

which is proven by induction on n :

Basic step: $n = 0$

$$\begin{aligned} & q \xrightarrow{\tau^0}_{\Xi} q' \\ \Rightarrow & (*q = q' \text{ and } q' \in Q_s *) \\ & q \xrightarrow{\tau^0} q' \end{aligned}$$

Induction step: $n = n' + 1$. We assume that equation 7 holds for n' .

$$\begin{aligned} & q \xrightarrow{\tau^{n'+1}}_{\Xi} q' \\ \Rightarrow & (*\text{Definition} \rightarrow *) \\ & \exists q_1 \in Q_{\Xi(s)} : q \xrightarrow{\tau^{n'}}_{\Xi} q_1 \wedge q_1 \xrightarrow{\tau}_{\Xi} q' \\ \Rightarrow & (*\text{Definition } \Xi, \text{ note that } q' \in Q_s *) \\ & \exists q_1 \in Q_s : q \xrightarrow{\tau^{n'}}_{\Xi} q_1 \wedge q_1 \xrightarrow{\tau} q' \\ \Rightarrow & (*\text{Induction } *) \\ & \exists q_1 \in Q_s : q \xrightarrow{\tau^{n'}} q_1 \wedge q_1 \xrightarrow{\tau} q' \\ \Rightarrow & (*\text{Definition} \rightarrow *) \\ & q \xrightarrow{\tau^{n'+1}} q' \end{aligned}$$

Induction step: $\sigma = \sigma' \cdot a$, where $a \in L_\delta$. We assume that the proposition holds for σ' . We identify two cases:

1. $a \in L$

$$\begin{aligned} & \Xi(s) \xrightarrow{\sigma' \cdot a}_{\Xi} q' \\ \Rightarrow & (*\text{Definition} \Rightarrow *) \\ & \exists q_1, q_2 \in Q_{\Xi(s)} : \Xi(s) \xrightarrow{\sigma'}_{\Xi} q_1 \wedge q_1 \xrightarrow{a}_{\Xi} q_2 \wedge q_2 \xrightarrow{\epsilon}_{\Xi} q' \\ \Rightarrow & (*\text{Equation 7, note that } q' \in Q_s *) \\ & \exists q_1 \in Q_{\Xi(s)}, q_2 \in Q_s : \Xi(s) \xrightarrow{\sigma'}_{\Xi} q_1 \wedge q_1 \xrightarrow{a}_{\Xi} q_2 \wedge q_2 \xrightarrow{\epsilon} q' \\ \Rightarrow & (*\text{Definition } \Xi, \text{ note that } q_2 \in Q_s *) \\ & \exists q_1, q_2 \in Q_s : \Xi(s) \xrightarrow{\sigma'}_{\Xi} q_1 \wedge q_1 \xrightarrow{a} q_2 \wedge q_2 \xrightarrow{\epsilon} q' \\ \Rightarrow & (*\text{Induction } *) \\ & \exists q_1, q_2 \in Q_s : s \xrightarrow{\sigma'} q_1 \wedge q_1 \xrightarrow{a} q_2 \wedge q_2 \xrightarrow{\epsilon} q' \\ \Rightarrow & (*\text{Definition} \Rightarrow *) \\ & s \xrightarrow{\sigma' \cdot a} q' \end{aligned}$$

$$\begin{aligned}
2. a = \delta & \\
& \Xi(s) \xrightarrow{\sigma' \cdot \delta} q' \\
& \Rightarrow (* \text{Definition } \Rightarrow *) \\
& \exists q_1, q_2 \in Q_{\Xi(s)} : s \xrightarrow{\sigma'} q_1 \wedge q_1 \xrightarrow{\delta} q_2 \wedge q_2 \xrightarrow{\epsilon} q' \\
& \Rightarrow (* \text{Definition of } \delta (q_1 = q_2 = q') *) \\
& \Xi(s) \xrightarrow{\sigma'} q' \wedge \forall \lambda \in U_{\tau} : q' \not\xrightarrow{\lambda} \\
& \Rightarrow (* \text{Induction } *) \\
& s \xrightarrow{\sigma'} q' \wedge \forall \lambda \in U_{\tau} : q' \not\xrightarrow{\lambda} \\
& \Rightarrow (* \text{Definition } \Xi *) \\
& s \xrightarrow{\sigma'} q' \wedge \forall \lambda \in U_{\tau} : q' \not\xrightarrow{\lambda} \\
& \Rightarrow (* \text{Definition } \delta *) \\
& s \xrightarrow{\sigma'} q' \wedge q' \xrightarrow{\delta} q' \\
& \Rightarrow (* \text{Definition of } \Rightarrow *) \\
& s \xrightarrow{\sigma' \cdot \delta} q'
\end{aligned}$$

Lemma A.9. Let q_X be a chaotic state in a demonically completed LTS:

$$\forall \sigma \in (U \cup \delta^* I)^* : q_X \xrightarrow{\sigma} q_X$$

Proof. Proof by induction on the structure of σ .

Basic step $\sigma = \epsilon$:

From the definition of $\Xi : q_X \xrightarrow{\tau^0} q_X$

Induction step : Suppose the lemma holds for σ' . We prove that it holds for $\sigma \in \sigma' \cdot (U \cup \delta^* I)$.

1. $\sigma = \sigma' \cdot \lambda$ where $\lambda \in U$

$$\begin{aligned}
& q_X \xrightarrow{\sigma'} q_X \\
& \Rightarrow (* \text{definition } \Xi : \forall \lambda \in U : q_X \xrightarrow{\tau} q_{\Omega} \xrightarrow{\lambda} q_X *) \\
& q_X \xrightarrow{\sigma'} q_X \xrightarrow{\lambda} q_X \\
& \Rightarrow (* \text{Definition } \Rightarrow *) \\
& q_X \xrightarrow{\sigma' \cdot \lambda} q_X \\
& \Rightarrow (* \sigma = \sigma' \cdot \lambda *)
\end{aligned}$$

2. $\sigma \in \sigma' \cdot \delta^* \cdot \lambda$ where $\lambda \in I$

$$\begin{aligned}
& q_X \xrightarrow{\sigma'} q_X \\
& \Rightarrow (* \text{definition } \Xi : \forall \lambda \in I : q_X \xrightarrow{\tau} q_{\Delta} \xrightarrow{\delta^*} q_{\Delta} \xrightarrow{\lambda} q_X *) \\
& q_X \xrightarrow{\sigma'} q_X \xrightarrow{\delta^* \cdot \lambda} q_X \\
& \Rightarrow (* \text{Definition } \Rightarrow *) \\
& q_X \xrightarrow{\sigma' \cdot \delta^* \cdot \lambda} q_X \\
& \Rightarrow (* \sigma \in \sigma' \cdot \delta^* \cdot \lambda *) \\
& q_X \xrightarrow{\sigma} q_X
\end{aligned}$$

Proposition 16 Let q_χ be a chaotic state in a demonically completed LTS:

$$\forall \sigma \in (U \cup \delta^* I)^* \delta^* : q_\chi \xrightarrow{\sigma} q' \wedge q' \notin Q_s$$

Proof. From lemma A.9 we already know that the lemma holds for traces in the regular set $(U \cup \delta^* I)^*$. Now we have to prove that the lemma holds if traces of this form end with δ^* . Let $\sigma \in (U \cup \delta^* I)^*$

$$\begin{aligned} & \sigma \in (U \cup \delta^* I)^* \\ \Rightarrow & (* \text{Lemma A.9 } *) \\ & q_\chi \xrightarrow{\sigma} q_\chi \\ \Rightarrow & (* \text{Definition } \Xi *) \\ & q_\chi \xrightarrow{\sigma} q_\chi \xrightarrow{\tau} q_\Delta \xrightarrow{\delta^*} q_\Delta \\ \Rightarrow & (* \text{Definition } \Rightarrow *) \\ & q_\chi \xrightarrow{\sigma \cdot \delta^*} q_\Delta \end{aligned}$$

We delay the proof of theorem 17 until after theorem 21.

Proposition 20 Let $s \in \mathcal{LTS}(I, U)$.

$$Utraces(s) = \{\sigma \in Straces(s) \mid (\Xi(s) \text{ after } \sigma) \subseteq Q_s\}$$

Proof. We prove the proposition in two steps:

1. $Utraces(s) \subseteq \{\sigma \in Straces(s) \mid (\Xi(s) \text{ after } \sigma) \subseteq Q_s\}$

We will prove this equation by reductio ad absurdum. Suppose the above proposition does not hold, then there exists a σ such that the following holds:

$$\sigma \in Utraces(s) \wedge (\sigma \notin Straces(s) \vee (\Xi(s) \text{ after } \sigma) \not\subseteq Q_s)$$

There are two possibilities:

1. $\sigma \in Utraces(s) \wedge \sigma \notin Straces(s)$. We prove that this gives rise to a contradiction.

$$\begin{aligned} & \sigma \in Utraces(s) \wedge \sigma \notin Straces(s) \\ \Rightarrow & (* \text{Definition } Straces *) \\ & \sigma \in Utraces(s) \wedge (\sigma \notin L_\delta^* \vee s \xrightarrow{\sigma}) \\ \Rightarrow & (* \text{Definition } Utraces *) \\ & \sigma \in L_\delta^* \wedge s \xrightarrow{\sigma} \wedge (\sigma \notin L_\delta^* \vee s \xrightarrow{\sigma}) \\ \Rightarrow & \text{Contradiction} \end{aligned}$$

2. $\sigma \in Utraces(s) \wedge (\Xi(s) \text{ after } \sigma) \not\subseteq Q_s$. We prove that this also gives rise to a contradiction.
- $\sigma \in Utraces(s) \wedge (\Xi(s) \text{ after } \sigma) \not\subseteq Q_s$
 - \Rightarrow (*Definition after *)
 - $\sigma \in Utraces(s) \wedge \{q \mid \Xi(s) \xrightarrow{\sigma} q\} \not\subseteq Q_s$
 - \Rightarrow (*Set definition *)
 - $\sigma \in Utraces(s) \wedge \exists q \notin Q_s : \Xi(s) \xrightarrow{\sigma} q$
 - \Rightarrow (*Definition Ξ , take the first time that $\Xi(s)$ enters a state not in Q_s *)
 - $\sigma \in Utraces(s)$
 - $\wedge \exists q \notin Q_s, q' \in Q_s, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma : \lambda \in I \wedge \Xi(s) \xrightarrow{\sigma_1} q' \wedge q' \xrightarrow{\lambda} q_X \xrightarrow{\sigma_2} q$
 - \Rightarrow (*Proposition 15 *)
 - $\sigma \in Utraces(s) \wedge \exists q' \in Q_s, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma : \lambda \in I \wedge s \xrightarrow{\sigma_1} q' \wedge q' \xrightarrow{\lambda} q_X$
 - \Rightarrow (*Definition Ξ *)
 - $\sigma \in Utraces(s)$
 - $\wedge \exists q' \in Q_s, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma : \lambda \in I \wedge s \xrightarrow{\sigma_1} q' \wedge q' \xrightarrow{\lambda} q' \not\rightarrow$
 - \Rightarrow (*Definition $\xrightarrow{\lambda}$ *)
 - $\sigma \in Utraces(s) \wedge \exists q' \in Q_s, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma : \lambda \in I \wedge s \xrightarrow{\sigma_1} q' \wedge q' \not\rightarrow^{\lambda}$
 - \Rightarrow (*Definition $Utraces$ *)
 - $(\sigma \in L_\delta^* \wedge s \xrightarrow{\sigma} \wedge \nexists q' \in Q_s, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma : \lambda \in I \wedge s \xrightarrow{\sigma_1} q' \wedge q' \not\rightarrow^{\lambda})$
 - $\wedge (\exists q' \in Q_s, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma : \lambda \in I \wedge s \xrightarrow{\sigma_1} q' \wedge q' \not\rightarrow^{\lambda})$
 - \Rightarrow Contradiction
3. $Utraces(s) \supseteq \{\sigma \in Straces(s) \mid (\Xi(s) \text{ after } \sigma) \subseteq Q_s\}$.
- We will prove this equation by reductio ad absurdum. Suppose that the above proposition does not hold, then the following holds:
- $\sigma \in Straces(s) \wedge \Xi(s) \text{ after } \sigma \subseteq Q_s \wedge \sigma \notin Utraces(s)$
 - \Rightarrow (*Definition $Utraces$ *)
 - $\sigma \in Straces(s) \wedge \Xi(s) \text{ after } \sigma \subseteq Q_s$
 - $\wedge (\sigma \notin L_\delta^* \vee s \not\rightarrow^{\sigma} \vee \exists (q_1, \sigma_1 \cdot \lambda \cdot \sigma_2) : \lambda \in I \wedge s \xrightarrow{\sigma_1} q_1 \wedge q_1 \not\rightarrow^{\lambda})$
 - \Rightarrow (*Definition $Straces$ *)
 - $\sigma \in L_\delta^* \wedge s \xrightarrow{\sigma} \wedge \Xi(s) \text{ after } \sigma \subseteq Q_s$
 - $\wedge (\sigma \notin L_\delta^* \vee s \not\rightarrow^{\sigma} \vee \exists (q_1, \sigma_1 \cdot \lambda \cdot \sigma_2) : \lambda \in I \wedge s \xrightarrow{\sigma_1} q_1 \wedge q_1 \not\rightarrow^{\lambda})$
- There are three possibilities in the last conjunct:
1. $\sigma \notin L_\delta^*$
 \Rightarrow Contradiction
 2. $s \not\rightarrow^{\sigma}$
 \Rightarrow Contradiction

$$\begin{aligned}
3. & \exists(q_1, \sigma_1 \cdot \lambda \cdot \sigma_2) : \lambda \in I \wedge s \xrightarrow{\sigma_1} q_1 \wedge q_1 \not\xrightarrow{\lambda} \\
& \Rightarrow (*s \text{ is strongly converging } *) \\
& \sigma \in L_\delta^* \wedge s \xrightarrow{\sigma} \wedge \Xi(s) \text{ after } \sigma \subseteq Q_s \\
& \wedge \exists(q_1, q_2, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma) : \lambda \in I \wedge s \xrightarrow{\sigma_1} q_1 \wedge q_1 \xrightarrow{\epsilon} q_2 \wedge q_2 \not\xrightarrow{\lambda} \wedge q_2 \not\xrightarrow{\tau} \\
& \Rightarrow (*\text{Definition } \xrightarrow{\sigma_1} *) \\
& \Xi(s) \text{ after } \sigma \subseteq Q_s \\
& \wedge \exists(q_2, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma) : \lambda \in I \wedge s \xrightarrow{\sigma_1} q_2 \wedge q_2 \not\xrightarrow{\lambda} \wedge q_2 \not\xrightarrow{\tau} \\
& \Rightarrow (*\text{Proposition 15 } *) \\
& \Xi(s) \text{ after } \sigma \subseteq Q_s \\
& \wedge \exists(q_2, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma) : \lambda \in I \wedge \Xi(s) \xrightarrow{\sigma_1} q_2 \wedge q_2 \not\xrightarrow{\lambda} \wedge q_2 \not\xrightarrow{\tau} \\
& \Rightarrow (*\text{Definition } \Xi *) \\
& \Xi(s) \text{ after } \sigma \subseteq Q_s \\
& \wedge \exists(q_2, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma) : \lambda \in I \wedge \Xi(s) \xrightarrow{\sigma_1} q_2 \wedge q_2 \xrightarrow{\lambda} q_\chi \\
& \Rightarrow (*\text{Proposition 14 } *) \\
& \Xi(s) \text{ after } \sigma \subseteq Q_s \\
& \wedge \sigma \in (U \cup \delta^* I)^* \delta^* \wedge \exists(q_2, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma) : \lambda \in I \wedge \Xi(s) \xrightarrow{\sigma_1} q_2 \wedge q_2 \xrightarrow{\lambda} q_\chi \\
& \Rightarrow (*\text{Lemma A.8 } *) \\
& \Xi(s) \text{ after } \sigma \subseteq Q_s \wedge \sigma \in (U \cup \delta^* I)^* \delta^* \\
& \wedge \exists(q_2, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma) : \sigma_2 \in (U \cup \delta^* I)^* \delta^* \wedge \lambda \in I \wedge \Xi(s) \xrightarrow{\sigma_1} q_2 \wedge q_2 \xrightarrow{\lambda} q_\chi \\
& \Rightarrow (*\text{Proposition 16 } *) \\
& \Xi(s) \text{ after } \sigma \subseteq Q_s \wedge \exists(q \notin Q_s, \sigma_1 \cdot \lambda \cdot \sigma_2 = \sigma) : \Xi(s) \xrightarrow{\sigma_1} q_2 \xrightarrow{\lambda} q_\chi \xrightarrow{\sigma_2} q \\
& \Rightarrow (*\text{Definition } \subseteq *) \\
& \Xi(s) \text{ after } \sigma \subseteq Q_s \wedge \{q \mid \Xi(s) \xrightarrow{\sigma} q\} \not\subseteq Q_s \\
& \Rightarrow (*\text{Definition after } *) \\
& \Xi(s) \text{ after } \sigma \subseteq Q_s \wedge \Xi(s) \text{ after } \sigma \not\subseteq Q_s \\
& \Rightarrow \text{Contradiction}
\end{aligned}$$

Lemma A.10. $\forall \sigma \in L_\delta^* : out(s \text{ after } \sigma) \subseteq out(\Xi(s) \text{ after } \sigma)$

Proof. Let $\sigma \in L_\delta^*$.

$$\begin{aligned}
& out(s \text{ after } \sigma) \\
& = (*\text{Definition after } *) \\
& out(\{q' \mid s \xrightarrow{\sigma} q'\}) \\
& = (*\text{Definition out } *) \\
& \{y \in U \mid s \xrightarrow{\sigma} q' \xrightarrow{y} \} \cup \{\delta \mid s \xrightarrow{\sigma} q' \xrightarrow{\delta} \} \\
& \subseteq (*\text{Proposition 15 } *) \\
& \{y \in U \mid \Xi(s) \xrightarrow{\sigma} q' \xrightarrow{y} \} \cup \{\delta \mid s \xrightarrow{\sigma} q' \xrightarrow{\delta} \} \\
& = (*\text{Definition out } *) \\
& out(\{q' \mid \Xi(s) \xrightarrow{\sigma} q'\}) \\
& = (*\text{Definition after } *) \\
& out(\Xi(s) \text{ after } \sigma)
\end{aligned}$$

Lemma A.11. $\forall \sigma \in Utraces(s) : out(s \text{ after } \sigma) = out(\Xi(s) \text{ after } \sigma)$

Proof.

$$\begin{aligned}
& out(\Xi(s) \text{ after } \sigma) \\
= & (*\text{Definition after and proposition 20 } *) \\
& out(\{q' \in Q_s \mid \Xi(s) \xrightarrow{\sigma}_{\Xi} q'\}) \\
= & (*\text{Definition out } *) \\
& \{y \in U \mid \exists q' \in Q_s : \Xi(s) \xrightarrow{\sigma}_{\Xi} q' \xrightarrow{y}_{\Xi}\} \cup \{\delta \mid \exists q' \in Q_s : \Xi(s) \xrightarrow{\sigma}_{\Xi} q' \xrightarrow{\delta}_{\Xi}\} \\
= & (*\text{Definition } \Xi, \text{ note that } y \in U \text{ and } q' \in Q_s *) \\
& \{y \in U \mid \exists q' \in Q_s : \Xi(s) \xrightarrow{\sigma}_{\Xi} q' \xrightarrow{y}\} \cup \{\delta \mid \exists q' \in Q_s : \Xi(s) \xrightarrow{\sigma}_{\Xi} q' \xrightarrow{\delta}\} \\
= & (*\text{Definition } \Xi \text{ and definition } \delta *) \\
& \{y \in U \mid \exists q' \in Q_s : \Xi(s) \xrightarrow{\sigma}_{\Xi} q' \xrightarrow{y}\} \cup \{\delta \mid \exists q' \in Q_s : \Xi(s) \xrightarrow{\sigma}_{\Xi} q' \xrightarrow{\delta}\} \\
= & (*\text{Proposition 15 } *) \\
& \{y \in U \mid s \xrightarrow{\sigma} q' \xrightarrow{y}\} \cup \{\delta \mid s \xrightarrow{\sigma} q' \xrightarrow{\delta}\} \\
= & (*\text{Definition out } *) \\
& out(\{q' \mid s \xrightarrow{\sigma} q'\}) \\
= & (*\text{Definition after } *) \\
& out(s \text{ after } \sigma)
\end{aligned}$$

Lemma A.12. *Let $s \in \mathcal{LTS}(I, U)$, $\sigma \in L_{\delta}^*$: $out(s \text{ after } \sigma \cdot \delta) \subseteq \{\delta\}$*

Proof.

$$\begin{aligned}
& out(s \text{ after } \sigma \cdot \delta) \\
= & (*\text{Definition after } *) \\
& out(\{q' \mid s \xrightarrow{\sigma \cdot \delta} q'\}) \\
= & (*\text{Definition } \delta *) \\
& out(\{q' \mid s \xrightarrow{\sigma} q', \text{ where } \forall y \in U_{\tau} : q' \not\xrightarrow{y}\}) \\
= & (*\text{Definition out } *) \\
& \{y \in U \mid s \xrightarrow{\sigma} q' \xrightarrow{y}, \text{ where } \forall y \in U_{\tau} : q' \not\xrightarrow{y}\} \\
& \cup \{\delta \mid s \xrightarrow{\sigma} q' \wedge \forall y \in U_{\tau} : q' \not\xrightarrow{y}, \text{ where } \forall y \in U_{\tau} : q' \not\xrightarrow{y}\} \\
\subseteq & (*\text{Logical reasoning; } \{\delta\} \text{ is the maximal outset } *) \\
& \{\delta\}
\end{aligned}$$

Lemma A.13. $\forall s \in \mathcal{LTS}(I, U) : Straces(s) \subseteq Straces(\Xi(s))$

Proof.

$$\begin{aligned}
& \sigma \in Straces(s) \\
\Rightarrow & (*\text{Definition Straces } *) \\
& \sigma \in L_{\delta}^* \wedge \exists q' : s \xrightarrow{\sigma} q' \\
\Rightarrow & (*\text{Proposition 15 } *) \\
& \sigma \in L_{\delta}^* \wedge \exists q' \in Q_s : \Xi(s) \xrightarrow{\sigma}_{\Xi} q' \\
\Rightarrow & (*\text{Definition Straces } *) \\
& \sigma \in Straces(\Xi(s))
\end{aligned}$$

Lemma A.14. $\forall s \in \mathcal{LTS}(I, U) : Utraces(s) \subseteq Straces(\Xi(s))$

Proof. The proof follows from proposition 20 and lemma A.13

Theorem 21

$$\mathbf{ioco}_U = \mathbf{ioco} \circ \Xi$$

Proof. Let $i \in \mathcal{IOTS}(I, U)$, $s \in \mathcal{LTS}(I, U)$ To make the proof easier to read we first expand the **ioco** definition:

$$\begin{aligned} \forall \sigma \in \text{Utraces}(s) : \text{out}(i \textbf{ after } \sigma) \subseteq \text{out}(s \textbf{ after } \sigma) \\ \Leftrightarrow \forall \sigma \in \text{Straces}(\Xi(s)) : \text{out}(i \textbf{ after } \sigma) \subseteq \text{out}(\Xi(s) \textbf{ after } \sigma) \end{aligned}$$

only if: Take $\sigma \in \text{Straces}(\Xi(s))$ arbitrary. There are two possibilities:

1. $\sigma \in \text{Straces}(\Xi(s)) \cap \text{Utraces}(s)$
 - \Rightarrow (*Lemma A.14 *)
 - $\sigma \in \text{Utraces}(s)$
 - \Rightarrow (*Premise *)
 - $\sigma \in \text{Utraces}(s) \wedge \text{out}(i \textbf{ after } \sigma) \subseteq \text{out}(s \textbf{ after } \sigma)$
 - \Rightarrow (*Lemma A.10 *)
 - $\sigma \in \text{Utraces}(s) \wedge \text{out}(i \textbf{ after } \sigma) \subseteq \text{out}(\Xi(s) \textbf{ after } \sigma)$
2. $\sigma \in \text{Straces}(\Xi(s)) \setminus \text{Utraces}(s)$
 - \Rightarrow (*Proposition 20 *)
 - $\sigma \in \text{Straces}(\Xi(s)) \wedge (\sigma \notin \text{Straces}(s) \vee \Xi(s) \textbf{ after } \sigma \not\subseteq Q_s)$
 - \Rightarrow (*Definition Straces *)
 - $\exists q \in Q_{\Xi(s)} : \Xi(s) \xrightarrow{\sigma} q \wedge (\nexists q \in Q_s : s \xrightarrow{\sigma} q \vee \Xi(s) \textbf{ after } \sigma \not\subseteq Q_s)$
 - \Rightarrow (*Logical reasoning *)
 - $\exists q \in Q_{\Xi(s)} : \Xi(s) \xrightarrow{\sigma} q \wedge \nexists q \in Q_s : s \xrightarrow{\sigma} q$
 - $\vee \exists q \in Q_{\Xi(s)} : \Xi(s) \xrightarrow{\sigma} q \wedge \Xi(s) \textbf{ after } \sigma \not\subseteq Q_s$
 - \Rightarrow (*Definition after *)
 - $\exists q \in Q_{\Xi(s)} : \Xi(s) \xrightarrow{\sigma} q \wedge \nexists q \in Q_s : s \xrightarrow{\sigma} q$
 - $\vee \exists q \in Q_{\Xi(s)} : \Xi(s) \xrightarrow{\sigma} q \wedge \exists q \in Q_{\Xi(s)} \setminus Q_s : \Xi(s) \xrightarrow{\sigma} q$
 - \Rightarrow (*Logical reasoning *)
 - $\exists q \in Q_{\Xi(s)} \setminus Q_s : \Xi(s) \xrightarrow{\sigma} q$
 - \Rightarrow (*Definition Ξ *)
 - $\exists q \in Q_{\Xi(s)} \setminus Q_s, \sigma_1 \cdot \sigma_2 = \sigma : \Xi(s) \xrightarrow{\sigma_1} q_X \xrightarrow{\sigma_2} q$

Using Proposition 14 and Lemma A.8 we have $\sigma \in (U \cup \delta^* I)^*$. There are two possibilities left:

1. $\sigma_2 \in (U \cup \delta^* I)^*$ (trace does not end with δ). The proof continues as follows:
 - \Rightarrow (*Lemma A.9 *)
 - $\exists \sigma_1 \cdot \sigma_2 = \sigma : \Xi(s) \xrightarrow{\sigma_1} q_X \xrightarrow{\sigma_2} q_X$
 - \Rightarrow (*Definition Ξ *)
 - $\{q \mid \Xi(s) \xrightarrow{\sigma} q\} \supseteq \{q_X, q_\Omega, q_\Delta\}$
 - \Rightarrow (*Definition after *)
 - $\Xi(s) \textbf{ after } \sigma \supseteq \{q_X, q_\Omega, q_\Delta\}$
 - \Rightarrow (*Proposition 16 *)
 - $\text{out}(\Xi(s) \textbf{ after } \sigma) = U_\delta$
 - \Rightarrow (* U_δ is the maximal outset *)
 - $\forall i \in \mathcal{IOTS}(I, U) : \text{out}(i \textbf{ after } \sigma) \subseteq \text{out}(\Xi(s) \textbf{ after } \sigma)$

2. $\sigma_2 \in (U \cup \delta^* I)^* \delta \delta^*$ (trace ends with δ). The proof continues as follows:
 \Rightarrow (*Premise: $\sigma \in \text{Straces}(\Xi(s))$ *)
 $\exists q \in Q_{\Xi(s)} : \Xi(s) \xrightarrow{\sigma} q \xrightarrow{\delta}$
 \Rightarrow (*Definition *out* and **after** *)
 $\text{out}(\Xi(s) \mathbf{after} \sigma) = \{\delta\}$
 \Rightarrow (*Lemma A.12 *)
 $\text{out}(i \mathbf{after} \sigma) \subseteq \text{out}(\Xi(s) \mathbf{after} \sigma)$

if:

$\forall \sigma \in \text{Straces}(\Xi(s)) : \text{out}(i \mathbf{after} \sigma) \subseteq \text{out}(\Xi(s) \mathbf{after} \sigma)$
 \Rightarrow (*Lemma A.14 *)
 $\forall \sigma \in \text{Utraces}(s) : \text{out}(i \mathbf{after} \sigma) \subseteq \text{out}(\Xi(s) \mathbf{after} \sigma)$
 \Rightarrow (*Lemma A.11 *)
 $\forall \sigma \in \text{Utraces}(s) : \text{out}(i \mathbf{after} \sigma) \subseteq \text{out}(s \mathbf{after} \sigma)$

Theorem 17 Let $i \in \mathcal{IOTS}(I, U)$, $s \in \mathcal{LTS}(I, U)$ then

$$i \mathbf{ioco} s \Rightarrow i(\mathbf{ioco} \circ \Xi)s$$

Proof. This proposition follows from theorem 21.

References

- [1] A. Belinfante, J. Feenstra, R. d. Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink. Formal test automation: A simple experiment. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, *12th Int. Workshop on Testing of Communicating Systems*, pages 179 – 196. Kluwer Academic Publishers, 1999.
- [2] 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.
- [3] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the Association for Computing Machinery*, 31(3):560–599, July 1984.
- [4] R. De Nicola and R. Segala. A process algebraic view of Input/Output Automata. *Theoretical Computer Science*, 138:391–423, 1995.
- [5] J.-C. Fernandez, C. Jard, T. Jérón, and C. Viho. An experiment in automatic generation of test suites for protocols with verification technology. *Science of Computer Programming – Special Issue on COST247, Verification and Validation Methods for Formal Descriptions*, 29(1–2):123–146, 1997.
- [6] J. He and K. Turner. Protocol-Inspired Hardware Testing. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, *Int. Workshop on Testing of Communicating Systems 12*, pages 131–147. Kluwer Academic Publishers, 1999.
- [7] 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.
- [8] C. Jard, T. Jérón, L. Tanguy, and C. Viho. Remote testing can be as powerful as local testing. In *Formal Description Techniques and Protocol Specification, Testing and Verification FORTE XI /PSTV XVIII '99*. Kluwer Academic Publishers, 1999.

- [9] A. Petrenko and N. Yevtushenko. Fault detection in embedded components. In M. Kim, S. Kang, and K. Hong, editors, *Tenth Int. Workshop on Testing of Communicating Systems*, pages 272–287. Chapman & Hall, 1997.
- [10] A. Petrenko, N. Yevtushenko, and G. Von Bochman. Fault models for testing in context. In R. Gotzhein and J. Brederke, editors, *FORTE*, volume 69 of *IFIP Conference Proceedings*, pages 163 – 178. Kluwer, 1996.
- [11] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, 17(3):103–120, 1996.
- [12] J. Tretmans. Testing concurrent systems: A formal approach. In J. Baeten and S. Mauw, editors, *CONCUR'99*, volume 1664, pages 46 – 65. LNCS, Springer-Verlag, 1999.