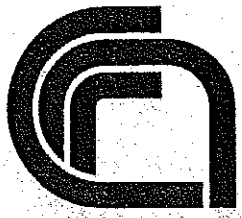

Consiglio Nazionale delle Ricerche

Relating testing and conformance relations for UML Statechart Diagrams Behaviours

Diego Latella Mieke Massink

Technical Report
CNUCE-B04-2002-001

CNUCE

Pisa

Relating Testing and Conformance Relations for UML Statechart Diagrams Behaviours*

Diego Latella Mieke Massink †

Abstract

In this paper we study the formal relationship between testing preorder/equivalences for a behavioural subset of UML Statechart Diagrams and a conformance relation for implementations with respect to specifications given using such diagrams. We study the impact of *stuttering* on the above mentioned relationship. In the context of UML Statechart Diagrams, *stuttering* coincides with absence of reaction of a machine on a given state and given input event. We consider both the case in which the semantics underlying the testing relations *does not* model stuttering explicitly - we call it the *non-stuttering* semantics - and the case in which it *does* it - *stuttering* semantics. We show that in the first case the conformance relation is stronger than the reverse of the MUST preorder and, consequently, stronger than the MAY preorder. Much more interesting results can be proven in the second case, possibly under proper conditions on the input set. In fact the conformance relation is shown to coincide with the MAY preorder, and thus be implied by the reverse MUST preorder. Finally, we show important substitutivity properties which hold in the case of stuttering semantics.

Keywords: UML Statechart Diagrams; Formal Semantics; Conformance Testing; Testing Theory.

1 Introduction

In this paper we study the formal relationship between testing preorder/equivalences for a behavioral subset of UML Statechart Diagrams (UMLSDs) and a conformance relation for implementations with respect to specifications given using such diagrams.

The fundamental condition underlying formal testing theory as well as formal conformance testing is the existence of formal semantics for the language at hand. We thoroughly discussed the benefits of a "semantics-first" based approach in [8, 3], where we also motivated our choice of a "kernel" of the notation and compared the semantics definition we proposed in [9, 4] with other proposals, e.g. [23, 2, 14, 13]. Our operational semantics maps every UMLSDs - actually an abstract syntax of the UMLSD, namely a hierarchical automaton (HA) - to a labelled transition system (LTS), where transition labels are input/out pairs (i/o-pairs). Each transition of the LTS corresponds to firing a maximal set of non-conflicting transitions of the UMLSD and is labelled with the input event which causes such transitions to be fired and the resulting (composite) output.

Testing and conformance relations in the context of LTSs have been thoroughly investigated in the literature. An account on the major results can be found in [6, 20, 21, 22]. To our knowledge, the study of such relations, and of testing in general, in the context of (non-deterministic) LTSs over i/o-pairs has received scant attention, instead. Exceptions are [12] and [1] where, on the other hand, only *deterministic* LTSs over i/o-pairs, actually Mealy machines, are considered. In [1] further restrictions on the LTSs are required, namely they must be strongly connected.

*This work has been carried out also under the Agreement between CNR CNUCE/IEI and GMD in the frame of the Project "Formal Test Cases Derivation for UML Statechart Diagrams Specifications".

†Consiglio Nazionale delle Ricerche, Area della Ricerca di Pisa, Istituto CNUCE, Via Alfieri 1, Loc. Ghezzano, I56010 S. Giuliano T. (PI), ITALY, email:{d.latella,m.massink}@cnuce.cnr.it

The basic notion underlying testing relations and theories is that systems are considered equivalent if and only if they cannot be distinguished by any "experimenter" interacting with them.

Testing relations for UMLSDs have been introduced in [10, 11] where a general testing theory for UMLSDs is proposed based on notions originally introduced in [6] and revisited in [16] in the context of systems with inputs and outputs. A Testing preorder is defined in the usual way as the intersection of proper MAY and MUST preorders. An algorithm for automatic verification of testing equivalence for UMLSDs has also been proposed in [11] and proved correct [10]. When no transition of the UMLSD is enabled to fire for given input event and given configuration, the semantics LTS has no transition from the corresponding state with such event as input label. Thus, in [10, 11], absence of reaction of the state machine is modeled by absence of a corresponding transition in the LTS. Lack of such transitions will result in deadlocks with experimenters. This feature is common in the process algebra framework [6]. In the sequel, we call such LTS the *non-stuttering* semantics of the UMLSD.

Broadly speaking, formal conformance testing refers to a field of theory, methodology and applications for testing that a given implementation of a system *conforms* to its abstract specification, where a proper conformance relation is defined using the formal semantics of the notation(s) used for defining the specification and modeling the implementation.

An interesting formal conformance relation in the context of labelled transition system models is the **ioco** relation [21], which has been successfully used in industrial contexts and for which efficient tools for the automatic generation of test cases from formal specifications, and automatic testing, have been produced [22].

A formal conformance relation for UMLSDs, namely $\sqsubseteq_{co}$, has been proposed in [5] based on concepts introduced in [21, 22]. Essentially, (a model of) an implementation conforms to (a model of) a specification if and only if the implementation can never produce an output, on a given input, which could not be produced by the specification "in the same situation", that is after the same sequence of steps and on the same input. Moreover, the implementation is allowed to produce no output at all, on a given input, only if the specification can do so. In [5] a *stuttering* semantics is used: when there are no transitions of the UMLSD enabled to fire for given input event and given configuration, the semantics LTS has a *stuttering* transition from the corresponding state, with such event as input label, and the special symbol Σ as output, explicitly indicating the absence of any reaction to such input event - actually such a transition is just a loop to the same state. The explicit representation of absence of output reaction is common practice in the framework of formal conformance testing. It is often referred to as "quiescence" and is an abstraction for techniques, like timeouts, for detecting, with high probability, lack of reaction at the system implementation level. An algorithm for automatic test case generation has also been proposed in [5], where its completeness - i.e. soundness and exhaustiveness - has been proved.

In this paper we show that $\sqsubseteq_{co}$ is strictly stronger than the reverse of the MUST preorder based on the non-stuttering semantics, which is well known to be stronger than the MAY preorder. Moreover, the Testing preorder and the conformance relation are incomparable; neither one is stronger than the other nor vice-versa. Furthermore, no substitutivity property holds: replacing an implementation conforming to a specification with a testing equivalent implementation may break conformance; symmetrically, an implementation conforming to a specification is not guaranteed to conform also to another, testing equivalent, specification.

All the above shows that using a non-stuttering semantics for testing equivalence is not a good choice when having conformance testing in mind. In fact, in conformance testing, the ability of detecting absence of reaction plays a major role. Consequently, experimenters must be allowed to detect absence of reaction and behave accordingly. In our framework, this means that we have to base the testing preorders on a stuttering semantics. In fact, in this case, the MAY preorder is stronger that $\sqsubseteq_{co}$, which implies that $\sqsubseteq_{co}$ contains the reverse of the MUST preorder. Thus the Testing preorder is stronger than the reverse conformance relation. This allows us to prove that important substitutivity properties like those mentioned above *do* hold in the stuttering semantics case.

The present paper is organized as follows. Sect. 2 shortly introduces basic notions and summarizes the non-stuttering and the stuttering semantics definitions. The formal links between such

semantics as well as them and our original one, which has been proven correct with respect to the official UML one [3] are also given. Sect. 3 briefly recalls main definitions and results for the testing theory for UMLSDs we use in this paper, while Sect. 4 recalls those for the conformance testing. Sect. 5 relates the Testing Preorders and $\sqsubseteq_{\mathsf{co}}$, presenting the main results of the present paper. Concluding remarks are given in Sect. 6 while all detailed proofs as well as background information are given in the appendices.

2 Hierarchical Automata

2.1 Basic definitions and Semantics

The first notion is that of a (sequential) automaton[1].

Definition 2.1 (Sequential Automata) *A sequential automaton A is a 4-tuple $(\sigma_A, s^0_A, \lambda_A, \delta_A)$ where σ_A is a finite set of* states *with $s^0_A \in \sigma_A$ the* initial state, *λ_A is a finite set of* transition labels *and $\delta_A \subseteq \sigma_A \times \lambda_A \times \sigma_A$ is the* transition relation. ∘

Hierarchical Automata are defined as follows:

Definition 2.2 (Hierarchical Automata) *A HA H is a tuple (F, E, ρ), where F is a finite set of sequential automata with mutually disjoint sets of states, i.e. $\forall A_1, A_2 \in F.\ \sigma_{A_1} \cap \sigma_{A_2} = \emptyset$ and E is a finite set of* events*; the* refinement function *$\rho : \bigcup_{A \in F} \sigma_A \mapsto 2^F$ imposes a tree structure to F, i.e. (i) there exists a unique root automaton $A_{root} \in F$ such that $A_{root} \notin \bigcup rng\ \rho$, (ii) every non-root automaton has exactly one ancestor state: $\bigcup rng\ \rho = F \setminus \{A_{root}\}$ and $\forall A \in F \setminus \{A_{root}\}.\ \exists_1 s \in \bigcup_{A' \in F \setminus \{A\}} \sigma_{A'}.\ A \in (\rho\ s)$ and (iii) there are no cycles: $\forall S \subseteq \bigcup_{A \in F} \sigma_A.\ \exists s \in S.\ S \cap \bigcup_{A \in \rho s} \sigma_A = \emptyset$.* ∘

In the sequel we implicitly make reference to a generic HA $H = (F, E, \rho)$. In the remainder of this section we will deal with the UML semantics of HAs keeping in mind our testing purposes.
A *configuration* denotes a global state of a HA, composed of local states of component sequential automata:

Definition 2.3 (Configurations) *A* configuration *of H is a set $C \subseteq \bigcup_{A \in F} \sigma_A$ such that (i) $\exists_1 s \in \sigma_{A_{root}}.\ s \in C$ and (ii) $\forall s, A.\ s \in C \wedge A \in \rho\ s \Rightarrow \exists_1 s' \in A.\ s' \in C$.* ∘

The set of all configurations of H is denoted by Conf_H, while C^0_H denotes its *initial* configuration, namely the configuration composed only by initial states.
The operational semantics of HAs is given in terms of Labelled Transition Systems (LTSs):

Definition 2.4 (LTSs) *A LTS is a tuple $(S, s^0, \mathcal{L}, \longrightarrow)$ where S is the* set of states *with $s^0 \in S$ being the* initial *state, $\mathcal{L}$ is the* set of (transition) labels *and $\longrightarrow \subseteq S \times \mathcal{L} \times S$ is the* transition relation *of the LTS.*

We will often refer to LTSs with label set $\mathcal{L}$ as LTSs *over* $\mathcal{L}$. For $(s, l, s') \in \longrightarrow$ we write $s \xrightarrow{l} s'$. The notation $s \xrightarrow{l}$ will be a shorthand for $\exists s'.\ s \xrightarrow{l} s'$. Moreover, we will often identify a LTS with its initial state and, implicitly, all the states reachable therefrom via the transition relation. Conversely, each state of the LTS will identify the LTS having this state as the initial one. The following definitions are standard in the context of LTSs.

[1] In the following we will freely use a functional-like notation in our definitions where: (i) currying will be used in function application, i.e. $f\ a_1\ a_2 \ldots\ a_n$ will be used instead of $f(a_1, a_2, \ldots, a_n)$ and function application will be considered left-associative; (ii) for function $f : X \mapsto Y$ and $Z \subseteq X$, $f\ Z = \{y \in Y \mid \exists x \in Z.\ y = fx\}$, $rng\ f$ denotes the *range* of f and $f_{|Z}$ is the restriction of f to Z. For set X, the set of finite, respectively infinite, sequences over X will be denoted by X^*, respectively X^∞, for $x \in X$ we let x denote also the sequence in X^* composed by the single element x, while for $\gamma, \gamma' \in X^*$ we let the juxtaposition $\gamma\gamma'$ of γ with γ' denote their concatenation; moreover for $\gamma \in X^* \cup X^\infty$, the i-th element of γ will be denoted by $\gamma_{\downarrow i}$, whenever defined. (iii) $\exists_1 x.\ P\ x$ means that there exists a *unique* x such that $P\ x$

Definition 2.5 (Auxiliary Definitions for LTSs) *For LTS s over $\mathcal{L}$, $l \in \mathcal{L}$, $\gamma \in \mathcal{L}^*$, $\gamma' \in \mathcal{L}^\infty$*

- *The transition relation $\xrightarrow{\gamma}$ over finite sequences is defined in the obvious way: (a) $s \xrightarrow{\epsilon} s$ and (b) if $s \xrightarrow{\gamma} s'$ and $s' \xrightarrow{l} s''$, then $s \xrightarrow{\gamma l} s''$*
- *$s \xrightarrow{\gamma'}$ iff there exists infinite sequence $\hat{s}$ of states such that $\hat{s}_{\downarrow 1} = s$ and for all $i \geq 1$ $\hat{s}_{\downarrow i} \xrightarrow{\gamma_{\downarrow i}} \hat{s}_{\downarrow i+1}$*
- *The* language *of s is the set $L\ s = \{\gamma \in \mathcal{L}^* \mid \exists s'.\ s \xrightarrow{\gamma} s'\}$;*
- *The* initial steps *of s is the set $S\ s = \{l \in \mathcal{L} \mid s \xrightarrow{l}\}$*
- *The* acceptance sets *of s* after *γ is the set $\mathcal{AS}\ s\ \gamma = \{S\ s' \mid s' \in s\ \mathtt{after}\ \gamma\}$*
- *The states of s* after *γ is the set $(s\ \mathtt{after}\ \gamma) = \{s' \mid s \xrightarrow{\gamma} s'\}$; for Z set of states we let $(Z\ \mathtt{after}\ \gamma)$ be the set $\bigcup_{s' \in Z}(s'\ \mathtt{after}\ \gamma)$*

In this paper we will use LTSs where the labels in $\mathcal{L}$ are i/o-pairs, i.e. $\mathcal{L} = \mathcal{I} \times \mathcal{U}$, for some $\mathcal{I}$ and $\mathcal{U}$. For such LTSs the following auxiliary definitions apply:

Definition 2.6 (Auxiliary Definitions for LTSs over i/o-pairs) *For LTS s over $\mathcal{L} = \mathcal{I} \times \mathcal{U}$, $i \in \mathcal{I}$*

- *For set Z of states over s, the* output *of Z on i is the set $(\mathtt{OUT}\ Z\ i) = \bigcup_{s' \in Z}\{u \in \mathcal{U} \mid s' \xrightarrow{(i,u)}\}$;*
- *For X finite set of finite subsets of $\mathcal{I} \times \mathcal{U}$*
  - *we let*
    $$mfs\ X = \bigcup_{x \in X}(mf\ x)$$
    where
    $mf\ x = \{y \in (func\ x) \mid \nexists y' \in (func\ x).\ y \subset y'\}$ *and*
    $func\ x = \{y \subseteq x \mid \forall (e_1, \mathcal{E}_1), (e_2, \mathcal{E}_2) \in y.\ e_1 = e_2 \Rightarrow \mathcal{E}_1 = \mathcal{E}_2\}$
  - *and $\mathbf{c}\ X$ be the smallest set such that*
    i) $X \subseteq \mathbf{c}\ X$
    ii) if $x_1, x_2 \in \mathbf{c}\ X$ then also $x_1 \cup x_2 \in \mathbf{c}\ X$
    iii) if $x_1, x_2 \in \mathbf{c}\ X$ and $x_1 \subseteq x \subseteq x_2$ if $x_1, x_2 \in \mathbf{c}\ X$ and $x_1 \subseteq x \subseteq x_2$ then also $x \in \mathbf{c}\ X$.
- *s is* input enabled *iff $\forall s' \in S, i \in \mathcal{I}.\ \exists u \in \mathcal{U}.\ s' \xrightarrow{(i,u)}$*

∘

Functions mfs and cls are used in testing theory respectively for representing (non-deterministic) i/o-relations as sets of functions and for computing a proper closure of sets of sets i/o-pairs [11]. We will often use the shorthand $(\mathtt{OUT}\ s\ \gamma\ i)$ for $(\mathtt{OUT}(s\ \mathtt{after}\ \gamma)\ i)$ with a bit of overloading of $\mathtt{OUT}$. The following lemma relates set $\mathtt{OUT}$ with the language of a LTS:

Lemma 2.1
For s finite LTS over $\mathcal{I} \times \mathcal{U}, i \in \mathcal{I}, u \in \mathcal{U}$ and $\gamma \in (\mathcal{I} \times \mathcal{U})^*$ the following holds:
$u \in (\mathtt{OUT}\ s\ \gamma\ i)$ iff $\gamma(i, u) \in (L\ s)$ □

The operational semantics of a HA is defined as a LTS, where transitions are characterized by the *step*-relation and they are labelled by i/o-pairs. The input component of the pair is a single event which represents the stimulus for the transition to fire while the output component is a collection of events which the HA returns to the environment as (part of) the reaction to the stimulus (the other part being represented by the configuration change).

In the context of UMLSDs, states are called *statuses*. In this paper we assume statuses being the same as configurations. Moreover, we use the words "state" and "status" as synonym. The current environment is instead modeled separately, as we shall see in Sect. 2.2. In any case, we need a tool for modeling different environment "policies", i.e. ways in which events are dispatched to the state machine. We do this by using an abstract data types notation.

We assume that for set D, Θ_D denotes the set of all structures of a certain kind (like FIFO queues, or multi-sets, or sets) over D and we assume to have basic operations for inserting and removing elements from such structures. In particular, for $\mathcal{D}, \mathcal{D}'$, etc. in Θ_D, and $d \in D$, $(add\ \mathcal{D}\ d)$ denotes the structure obtained by adding d to structure $\mathcal{D}$. Similarly, $(join\ \mathcal{D}\ \mathcal{D}')$ denotes the structure obtained by merging $\mathcal{D}$ with $\mathcal{D}'$. The predicate $is\_join_{j=1}^{n}\mathcal{D}_j\ \mathcal{J}$ states that $\mathcal{J}$ is a possible *join* of $\mathcal{D}_1 \ldots \mathcal{D}_n$ and it is a way for expressing non-deterministic merge of $\mathcal{D}_1 \ldots \mathcal{D}_n$. Moreover, by $(Sel\ \mathcal{D}\ d\ \mathcal{D}')$ we mean that $\mathcal{D}'$ is the structure resulting from selecting d from $\mathcal{D}$, the selection policy depending on the choice for the particular semantics. Finally, *nil* is the empty structure and given sequence $r \in D^*$, $(new\ r)$ is the structure containing the elements of r (again, the existence and nature of any relation among the elements of $(new\ r)$ depends on the semantics of the particular structure).

In the sequel, for set of events E we let $\mathrm{IOP}_E = E \times \Theta_E$ be the set of i/o-pairs over E. Moreover, for $\Sigma \notin E \cup \Theta_E$ we let ${}^{\Sigma}\Theta_E$ be the set $\Theta_E \cup \{\Sigma\}$ and ${}^{\Sigma}\mathrm{IOP}_E = E \times {}^{\Sigma}\Theta_E$. We let $\mathcal{E}$, resp. Γ, range over Θ_E, resp. ${}^{\Sigma}\Theta_E$.

In [10, 11] we used a *non-stuttering* semantics for HAs where the absence of output reaction of a HA (F, E, ρ) in configuration $\mathcal{C}$ on event $e \in E$ resulted in the *absence* of any step-transition with input label e from $\mathcal{C}$. This modeling choice was motivated by a similar choice made in [16, 6] in the context of testing theory.

In the context of conformance testing [21, 22], instead, the absence of output reaction is usually explicitly modeled by a special action, which, in the case of HAs can be naturally associated to stuttering. Thus, in [5] we defined a *stuttering* semantics for HAs, where stuttering is modeled by the special symbol Σ.

In the following we will briefly recall the two semantics and their relation with our original semantics, as proposed in [4, 3] and recalled in the Appendix. In this way, the compliance results of our semantics relative to major requirements of the official definition of the UML, as in [18] or version 1.4, are easily extended to the new semantics.

Definition 2.7 (Non-stuttering Operational semantics) *The non-stuttering operational semantics of an HA H is the LTS over* IOP_E, $(\mathbf{LTS}\ H)$, *defined as follows:* $(\mathbf{LTS}\ H) = (\mathrm{Conf}_H, \mathcal{C}_H^0, \mathrm{IOP}_E, \longrightarrow)$ *where the* step-*transition relation* $\longrightarrow$ *is the smallest relation induced by the following deduction system:*

$$\frac{e \in E,\ L \neq \emptyset,\ H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{L} (\mathcal{C}', \mathcal{E})}{\mathcal{C} \xrightarrow{(e,\mathcal{E})} \mathcal{C}'}$$

∘

Definition 2.8 (Stuttering Operational semantics) *The stuttering operational semantics of an HA H is the LTS over* ${}^{\Sigma}\mathrm{IOP}_E$, $({}^{\Sigma}\mathbf{LTS}\ H)$, *defined as follows:* $({}^{\Sigma}\mathbf{LTS}\ H) = (\mathrm{Conf}_H, \mathcal{C}_H^0, {}^{\Sigma}\mathrm{IOP}_E, \longrightarrow_{\Sigma})$ *where the* step-*transition relation* $\longrightarrow_{\Sigma}$ *is the smallest relation induced by the following deduction system:*

$$\frac{e \in E,\ L \neq \emptyset,\ H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{L} (\mathcal{C}', \mathcal{E})}{\mathcal{C} \xrightarrow{(e,\mathcal{E})}_{\Sigma} \mathcal{C}'} \quad (1)$$

$$\frac{e \in E,\ H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{\emptyset} (\mathcal{C}', \mathcal{E})}{\mathcal{C} \xrightarrow{(e,\Sigma)}_{\Sigma} \mathcal{C}'} \quad (2)$$

Notice that both (**LTS** H) and ($^{\Sigma}$**LTS** H) are *finite*: they have a finite number of states and a finite number of transitions. Furthermore, notice that both LTSs are defined on *the same* set of states, namely the set Conf_H of configurations of H. In the remainder of this paper we will use the notation $^{\Sigma}\mathcal{C}$ for configuration $\mathcal{C}$ when we want to emphasize it being a state of ($^{\Sigma}$**LTS** H), thus avoiding confusion about which LTS we are dealing with. Any transition of (**LTS** H) or ($^{\Sigma}$**LTS** H) denotes the result of firing a maximal set of non-conflicting transitions of the sequential automaton of H which respect priorities.
In both the deduction system of the stuttering semantics and that of the non-stuttering one we make use of an auxiliary relation, namely $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$. Such a relation, which is defined by the deduction system proposed in [9] and recalled in the Appendix as the *core* semantics, models labelled transitions of the semantics of HA A under specific constraints P related to transition priority. L is the set containing the transitions of the sequential automata of A which are selected to fire when the current configuration (resp. input) is $\mathcal{C}$ (resp. $\mathcal{E}$) and the firing of which brings to configuration (resp. output actions) $\mathcal{C}'$ (resp. $\mathcal{E}'$).

We refer the interested reader to [9, 8, 10, 11] for a detailed discussion on the core semantics.

2.2 Interaction with the environment

The above operational semantics abstract from the specific environment the HA is interacting with. In this section we recall the definitions related to the interaction of HAs with their *environment* from [10, 11, 5].

Definition 2.9 (Environments) *An* environment $\mathcal{O}$ *on event set* E *is a tuple* $(\omega_{\mathcal{O}}, \omega^0_{\mathcal{O}}, \iota_{\mathcal{O}}, \longrightarrow)$ *where* $\omega_{\mathcal{O}}$ *is a set of* output *states, with* $\omega^0_{\mathcal{O}} \in \omega_{\mathcal{O}}$ *being the* initial *(output) state,* $\iota_{\mathcal{O}} \subseteq {}^{\Sigma}\Theta_E \mapsto \omega_{\mathcal{O}}$ *is the set of* input *states, each input state being a* total *function from (event) structures* Θ_E *or the "stuttering" symbol* Σ *to output states* [2]. *Finally* $\longrightarrow \subseteq (\omega_{\mathcal{O}} \times E \times \iota_{\mathcal{O}}) \cup (\omega_{\mathcal{O}} \times \mathrm{SPECIAL} \times \omega_{\mathcal{O}})$ *is the* transition relation, *with* $E \cap \mathrm{SPECIAL} = \emptyset$ *and* $\tau \in \mathrm{SPECIAL}$. ∘

The above definition generalizes concepts proposed in [6, 16] and has been originally introduced for non-stuttering semantics in [10, 11], which the interested reader is referred to for motivations and discussion. The definition has been extended to cover stuttering semantics in [5].

In a similar way as for LTSs, we often identify an environment with its initial state and, implicitly, all the states reachable therefrom via the transition relation. Conversely, each (output) state of the environment will identify the environment having this state as the initial one.

The following definition captures the idea of letting a HA H interact with its environment:

Definition 2.10 (Interaction System) *For HA* $H = (F, E, \rho)$ *and environment* $\mathcal{O} = (\omega_{\mathcal{O}}, \omega^0_{\mathcal{O}}, \iota_{\mathcal{O}}, \longrightarrow)$ *on* E, $< \mathcal{O}, (^{\Sigma}\mathbf{LTS}\ H) >$ *is the interaction system* $(\omega_{\mathcal{O}} \times \mathrm{Conf}_H, \rightsquigarrow)$ *where the interaction relation* $\rightsquigarrow \subseteq (\omega_{\mathcal{O}} \times \mathrm{Conf}_H) \times (\omega_{\mathcal{O}} \times \mathrm{Conf}_H)$ *is the smallest relation induced by the deduction system below where for* $((\mathcal{O}, C), (\mathcal{O}', C')) \in \rightsquigarrow$ *we write* $\mathcal{O} \parallel C \rightsquigarrow \mathcal{O}' \parallel C'$

$$\frac{\mathcal{O} \xrightarrow{e} \mathcal{I}, C \xrightarrow{(e,\Gamma)}_{\Sigma} C'}{\mathcal{O} \parallel C \rightsquigarrow (\mathcal{I}\ \Gamma) \parallel C'} \qquad \frac{\mathcal{O} \xrightarrow{\tau} \mathcal{O}'}{\mathcal{O} \parallel C \rightsquigarrow \mathcal{O}' \parallel C}$$

Obviously the notion of interaction system can be extended to the case of a generic LTS over $^{\Sigma}\mathrm{IOP}_E$ and not only to those LTSs generated from HAs. In particular we will be interested on interaction systems involving implementations, as defined is section 2.4

[2] Notice that, for obvious reasons, in [10, 11] input states are defined as total functions in $\Theta_E \mapsto \omega_{\mathcal{O}}$.

$$
\begin{array}{l}
Queue(\mathcal{F} : \Theta E) := \\
(Sel\ \mathcal{F}\ f\ \mathcal{F}') \Rightarrow (f; \lambda X : \Theta_E.Queue(join\ \mathcal{F}'\ X)) \\
\\
\Sigma\text{Queue}(\mathcal{F} : \Theta_E) := \\
(Sel\ \mathcal{F}\ f\ \mathcal{F}') \Rightarrow (f; \lambda X : {}^{\Sigma}\Theta_E.\ ((X \neq \Sigma \Rightarrow \Sigma\text{Queue}(join\ \mathcal{F}'\ X)) + \\
\qquad\qquad\qquad\qquad\qquad\qquad (X = \Sigma \Rightarrow \Sigma\text{Queue}(\mathcal{F}'))))
\end{array}
$$

Figure 1: Definition of Queue($\mathcal{F}$) and ΣQueue($\mathcal{F}$)

2.3 Formal links among the semantics

In this section we first recall the correctness results which form the formal link between (a) the non-stuttering semantics [10] and our original one (Theorem 2.1, proved in [10]), and (b) the stuttering semantics [5] and our original one (Theorem 2.2, proved in [5]). The original semantics is considered in the form given in [4, 3] and recalled in the Appendix. Secondly, we take a closer look at some features of the stuttering semantics and the formal relation between such semantics and the non-stuttering one.

In the theorems below the special environments $Queue(\mathcal{F} : \Theta_E)$ and $\Sigma\text{Queue}(\mathcal{F} : \Theta_E)$ are used. Their definition is given in Fig. 1 using a notation which we introduced in [10, 5] and which is recalled in the Appendix. The transition relation of the original semantics is denoted by $\xrightarrow{L}$.

Theorem 2.1
For hierarchical automaton $H = (F, E, \rho), \mathcal{C}, \mathcal{C}' \in \text{Conf}_H$, $\mathcal{E}$, $\mathcal{E}'$, $\mathcal{E}'' \in (\Theta\ E)$, *the following holds:* $(Queue(\mathcal{E})\ ||\ \mathcal{C}) \rightsquigarrow (Queue(join\ \mathcal{E}''\ \mathcal{E}')\ ||\ \mathcal{C}')$ *iff there exists* non-empty $L \subseteq (\mathcal{T}\ H)$ *such that* $(\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', (join\ \mathcal{E}''\ \mathcal{E}'))$ ◇

Theorem 2.2
For hierarchical automaton $H = (F, E, \rho), \mathcal{C}, \mathcal{C}' \in \text{Conf}_H$, $\mathcal{E}$, $\mathcal{E}'$, $\mathcal{E}'' \in \Theta_E$, *the following holds:* $(\Sigma Queue(\mathcal{E})\ ||\ \mathcal{C}) \rightsquigarrow (\Sigma Queue(join\ \mathcal{E}''\ \mathcal{E}')\ ||\ \mathcal{C}')$ *iff there exists* $L \subseteq (\mathcal{T}\ H)$ *such that* $(\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', (join\ \mathcal{E}''\ \mathcal{E}'))$ ◇

The following result formally links the stuttering and non-stuttering LTSs of HAs:

Theorem 2.3
For all HAs $H = (F, E, \rho)$, $e \in E$ *and* $\mathcal{C} \in \text{Conf}_H$ *the following holds:*

i) $\forall \mathcal{C}' \in \text{Conf}_H, \mathcal{E} \in \Theta_E.\ (\mathcal{C} \xrightarrow{(e,\mathcal{E})} \mathcal{C}'$ iff $\mathcal{C} \xrightarrow{(e,\mathcal{E})}_{\Sigma} \mathcal{C}')$

ii) $(\nexists \mathcal{C}' \in \text{Conf}_H, \mathcal{E} \in \Theta_E.\ \mathcal{C} \xrightarrow{(e,\mathcal{E})} \mathcal{C}')$ iff $\mathcal{C} \xrightarrow{(e,\Sigma)}_{\Sigma} \mathcal{C}'$

So the three semantics generate the same step-transitions, except for stuttering. We remind here that a HA H stutters on input event e when there is *no* transition of *any* sequential automaton of H enabled by e in the current status. In other words, stuttering happens when the machine does not accept e in the current state. This refusal is modelled (a) *implicitly* in the non-stuttering semantics by not generating a step-transition at all and (b) *explicitly* in the stuttering semantics by producing Σ as output action in the step-transition. The original semantics simply generates transitions with the empty set as a label when stuttering. It is important to point out that in the stuttering semantics, the absence of reaction on a given input e generates stuttering - and is represented by Σ - *only* if $e \in E$. If $e \notin E$ then no transition at all is generated, in a similar way as in the non-stuttering semantics. For this reason, in this paper, the definition of a LTS always includes the explicit specification of the input set E, specially when we compare different LTSs on the basis of testing/conformance relations, as in the following sections. Of course, one can also fix set E in advance and then compare different LTSs over the same input set. Obviously all the

results we present in the present paper will be valid also in this case. Nevertheless, we prefer to prove them in the most general case, limiting possible constraints on the input sets to only those cases in which this is strictly necessary. The following is a useful corollary to Theorem 2.3:

Corollary 2.1 *For all* $C, C' \in \mathrm{Conf}_H, \gamma \in \mathrm{IOP}^*_E$ $C \xrightarrow{\gamma} C'$ iff ${}^{\Sigma}C \xrightarrow{\gamma}_{\Sigma} {}^{\Sigma}C'$

The following lemma shows some interesting features of the stuttering semantics.

Lemma 2.2
For HA $H = (F, E, \rho)$, *all* $C \in \mathrm{Conf}_H$, *and* $e \in E$ *the following holds:*

i) $\exists \Gamma \in {}^{\Sigma}\Theta_E, C' \in \mathrm{Conf}_H.\ C \xrightarrow{(e,\Gamma)}_{\Sigma} C'$

ii) $C \xrightarrow{(e,\Sigma)}_{\Sigma} C'$ *for some* $C' \in \mathrm{Conf}_H$ *implies* $C = C'$

iii) $C \xrightarrow{(e,\Sigma)}_{\Sigma} C$ *implies* $\nexists \mathcal{E} \in \Theta_E, C' \in \mathrm{Conf}_H.\ C \xrightarrow{(e,\mathcal{E})}_{\Sigma} C'$

□

We close this section with a lemma relating the languages of (**LTS** H) and (${}^{\Sigma}$**LTS** H), where we use the following

Definition 2.11 $(\gamma \setminus \Sigma)$ *For* $\gamma \in {}^{\Sigma}\mathrm{IOP}^*_E$ *we define* $\gamma \setminus \Sigma$ *as follows*

$$\gamma \setminus \Sigma = \begin{cases} \epsilon & \text{if } \gamma = \epsilon \\ \gamma' \setminus \Sigma & \text{if } \gamma = (e, \Sigma)\gamma', \text{ for some } e \in E, \gamma' \in {}^{\Sigma}\mathrm{IOP}^*_E \\ (e, \mathcal{E})(\gamma' \setminus \Sigma) & \text{if } \gamma = (e, \mathcal{E})\gamma', \text{ for some } e \in E, \mathcal{E} \in \Theta_E, \gamma' \in {}^{\Sigma}\mathrm{IOP}^*_E \end{cases}$$

∘

Lemma 2.3
For HA $H = (F, E, \rho)$, *all* $C, C' \in \mathrm{Conf}_H$, $\gamma \in {}^{\Sigma}\mathrm{IOP}^*_E$ *the following holds:*

i) ${}^{\Sigma}C \xrightarrow{\gamma}_{\Sigma} {}^{\Sigma}C'$ *implies* $C \xrightarrow{\gamma\setminus\Sigma} C'$

ii) $\gamma \in L\ {}^{\Sigma}C$ *implies* $\gamma \setminus \Sigma \in (L\ {}^{\Sigma}C) \cap (L\ C)$

iii) $(L\ C) \subseteq (L\ {}^{\Sigma}C)$

□

2.4 Pragmatics

In the context of the present work, we will assume that a *specification* of system behavior is given in the form of a UMLSD, which will be represented by a HA $H = (F, E, \rho)$. Actually, we will identify H with (${}^{\Sigma}$**LTS** H) and we will be concerned only with the latter. An *implementation* for H will be modeled by an *input enabled* LTS over ${}^{\Sigma}\mathrm{IOP}_{E'}$ for some E' (not necessarily equal to E). Under the above assumptions, for simplicity, we will often speak of specifications over ${}^{\Sigma}\mathrm{IOP}_E$ and implementations over ${}^{\Sigma}\mathrm{IOP}_{E'}$, or simply specifications/implementations over E/E'. Notice that we do not require the LTS modeling implementations be necessarily generated from HAs. Any such a model can be obtained by any means, obviously including, but not limited to, the case in which the implementation is itself a HA (i.e. a UMLSD).

The above assumptions are quite standard in the context of formal testing and conformance theory and its applications.

3 Testing Relations

In the following we shall recall the main definitions and results concerning testing relations for HAs. They are based on the non-stuttering semantics and are dealt with in detail in [10, 11] where the interested reader is referred to.

Definition 3.1 (Experimenter) *An* experimenter $\mathcal{O}$ *on* E *is an environment* $(\omega_{\mathcal{O}}, \omega^0_{\mathcal{O}}, \iota_{\mathcal{O}}, \longrightarrow)$ *on* E *where* SPECIAL $= \{\tau, \mathbf{W}\}$. ∘

Definition 3.2 (Experimental System) *For HA* $H = (F, E, \rho)$ *and experimenter* $\mathcal{O} = (\omega_{\mathcal{O}}, \omega^0_{\mathcal{O}}, \iota_{\mathcal{O}}, \longrightarrow)$ *on* E, $< \mathcal{O}, (\mathbf{LTS}\ H) >$ *is the experimental system* $(\omega_{\mathcal{O}} \times \mathrm{Conf}_H, \rightsquigarrow, \mathrm{Success})$ *where* $(\omega_{\mathcal{O}} \times \mathrm{Conf}_H, \rightsquigarrow)$ *is an interaction system and the* Success *set is the set* $\{\mathcal{O}' \in \omega_{\mathcal{O}} \mid \exists \mathcal{O}'' \in \omega_{\mathcal{O}}.\ \mathcal{O}' \xrightarrow{\mathbf{W}} \mathcal{O}''\}$ ∘

Definition 3.3 (Computations) *A* computation *for experimental system* $< \mathcal{O}, (\mathbf{LTS}\ H) >$ *is a sequence of the form:*

$$\mathcal{O}_0 \,||\, \mathcal{C}_0 \rightsquigarrow \mathcal{O}_1 \,||\, \mathcal{C}_1 \rightsquigarrow \mathcal{O}_2 \,||\, \mathcal{C}_2 \rightsquigarrow \ldots \mathcal{O}_k \,||\, \mathcal{C}_k \rightsquigarrow \ldots$$

which is maximal, *i.e. either it is infinite or it is finite with terminal element* $\mathcal{O}_n \,||\, \mathcal{C}_n$ *which has the property that* $\mathcal{O}_n \,||\, \mathcal{C}_n \rightsquigarrow \mathcal{O}' \,||\, \mathcal{C}'$ *for no pair* $\mathcal{O}'$, $\mathcal{C}'$. $\mathcal{O}_0$ *and* $\mathcal{C}_0$ *are the initial states of* $\mathcal{O}$ *and* $(\mathbf{LTS}\ H)$. ∘

We let $\mathrm{Comp}(\mathcal{O}, \mathcal{C})$ denote the set of all computations whose initial element is $\mathcal{O} \,||\, \mathcal{C}$. From the definition of interaction system we know that every computation $\mathbf{C} \in \mathrm{Comp}(\mathcal{O}, \mathcal{C})$ gives raise to a derivation $\mathcal{C} \xrightarrow{\gamma}$ on the side of the LTS. In this case, we say that $\mathbf{C}$ *runs over* (**r.o.**) γ. Notice that different computations can run over the same i/o-sequence, while each computation uniquely characterizes an i/o-sequence.

Definition 3.4 (Successful Computations) *A computation is* successful *iff* $\mathcal{O}_n \in$ Success *for some* $n \geq 0$. ∘

In the following we shall make reference to $\mathcal{C} = (\mathbf{LTS}\ H)$ over IOP_E and $\mathcal{C}' = (\mathbf{LTS}\ H')$ over $\mathrm{IOP}_{E'}$ for some HAs H and H'. All notions introduced for LTSs should be intended to apply to the HAs the behavior of which those LTSs model, and so, ultimately, to the UMLSDs represented by such HAs.

Definition 3.5 (Testing Equivalence) *Let* $\mathrm{Result}(\mathcal{O}, \mathcal{C}) \subseteq \{\top, \bot\}$ *be defined by*

$\top \in \mathrm{Result}(\mathcal{O}, \mathcal{C})$ *iff* $\mathrm{Comp}(\mathcal{O}, \mathcal{C})$ *contains a successful computation*

$\bot \in \mathrm{Result}(\mathcal{O}, \mathcal{C})$ *iff* $\mathrm{Comp}(\mathcal{O}, \mathcal{C})$ *contains an unsuccessful computation*

We say that LTSs $\mathcal{C}$ *and* $\mathcal{C}'$ *are* testing equivalent, *written* $\mathcal{C} \sim \mathcal{C}'$, *iff for all experimenters* $\mathcal{O}$ *on* $E \cup E'$ $\mathrm{Result}(\mathcal{O}, \mathcal{C}) = \mathrm{Result}(\mathcal{O}, \mathcal{C}')$ ∘

Definition 3.6 (Testing Preorders) *For* $\mathcal{C}$ *LTSs over* IOP_E, $\mathcal{C}'$ *LTSs over* $\mathrm{IOP}_{E'}$ *we let*

i) $\mathcal{C} \sqsubseteq_{\sim_{MAY}} \mathcal{C}'$ *iff for every* $\mathcal{O}$ *on* $E \cup E'$, *if* $\top \in \mathrm{Result}(\mathcal{O}, \mathcal{C})$ *then* $\top \in \mathrm{Result}(\mathcal{O}, \mathcal{C}')$

ii) $\mathcal{C} \sqsubseteq_{\sim_{MUST}} \mathcal{C}'$ *iff for every* $\mathcal{O}$ *on* $E \cup E'$, *if* $\bot \notin \mathrm{Result}(\mathcal{O}, \mathcal{C})$ *then* $\bot \notin \mathrm{Result}(\mathcal{O}, \mathcal{C}')$

iii) $\mathcal{C} \sqsubseteq_{\sim} \mathcal{C}'$ *iff* $(\mathcal{C} \sqsubseteq_{\sim_{MAY}} \mathcal{C}') \wedge (\mathcal{C} \sqsubseteq_{\sim_{MUST}} \mathcal{C}')$ ∘

Lemma 3.1
For $\mathcal{C}$ *LTSs over* IOP_E *and* $\mathcal{C}'$ *LTSs over* $\mathrm{IOP}_{E'}$ $\mathcal{C} \sim \mathcal{C}'$ *iff* $\mathcal{C} \approx \mathcal{C}'$ □

The following definition gives and alternative, intentional, characterization of testing preorders:

Definition 3.7 (Alternative Characterization of Testing Preorders) *For finite LTSs $\mathcal{C}$ over* IOP_E *and $\mathcal{C}'$ over* $\mathrm{IOP}_{E'}$

i) $\mathcal{C} <<_{MAY} \mathcal{C}'$ *iff* $(L\ \mathcal{C}) \subseteq (L\ \mathcal{C}')$

ii) $\mathcal{C} <<_{MUST} \mathcal{C}'$ *iff* $\forall \gamma \in \mathrm{IOP}^*_{E \cup E'}.\ mfs(\mathbf{c}\ (\mathcal{AS}\ \mathcal{C}'\ \gamma)) \subset\subset mfs(\mathbf{c}\ (\mathcal{AS}\ \mathcal{C}\ \gamma))$

iii) $\mathcal{C} << \mathcal{C}'$ *iff* $\mathcal{C} <<_{MAY} \mathcal{C}' \wedge \mathcal{C} <<_{MUST} \mathcal{C}'$

where $X \subset\subset Y$ *iff* $\forall x \in X.\ \exists y \in Y.\ y \subseteq x$ ∘

Lemma 3.2
For $\mathcal{C}$ LTSs over IOP_E *and $\mathcal{C}'$ LTSs over* $\mathrm{IOP}_{E'}$ *the following holds:*
$\mathcal{C} \sqsubseteq_{\sim MUST} \mathcal{C}'$ *implies* $(L\ \mathcal{C}') \subseteq (L\ \mathcal{C})$ □

Theorem 3.1 *For all finite LTSs $\mathcal{C}$ over* IOP_E, *$\mathcal{C}'$ over* $\mathrm{IOP}_{E'}$ *the following holds:*

a) $\mathcal{C} \sqsubseteq_{\sim MAY} \mathcal{C}'$ *iff* $\mathcal{C} <<_{MAY} \mathcal{C}'$

b) $\mathcal{C} \sqsubseteq_{\sim MUST} \mathcal{C}'$ *iff* $\mathcal{C} <<_{MUST} \mathcal{C}'$

c) $\mathcal{C} \sqsubseteq_{\sim} \mathcal{C}'$ *iff* $\mathcal{C} << \mathcal{C}'$

□

4 Conformance Testing

Definition 4.1 (Conformance Relation) *For s' LTS over $\mathcal{I}' \times \mathcal{U}'$, s LTS over $\mathcal{I} \times \mathcal{U}$: $s' \sqsubseteq_{\mathsf{co}} s$ iff* $\forall \gamma \in (L\ s), i \in \mathcal{I}.\ \mathtt{OUT}\ s'\ \gamma\ i \subseteq \mathtt{OUT}\ s\ \gamma\ i$ ∘

Intuitively, $s' \sqsubseteq_{\mathsf{co}} s$ means that s' can never produce an output which could not be produced by s in the same situation, i.e. after the same i/o sequence *and the same input.* We will often refer to s' as an implementation and to s as a specification for s'. In general it is not required $\mathcal{I} = \mathcal{I}'$: for partial specifications $\mathcal{I} \subseteq \mathcal{I}'$ will generally hold, while for incomplete implementations $\mathcal{I}' \subseteq \mathcal{I}$ will hold; $\mathcal{I} \cap \mathcal{I}' = \emptyset$ does not make much sense. Notice that, for $\mathcal{U} = {}^{\Sigma}\Theta_E$ and $\mathcal{U}' = {}^{\Sigma}\Theta_{E'}$ the above definition implies that s' may produce no output at all only if s can do so. Thus, this is the same concept as in [21, 22] but its technical definition is adapted to specifications over ${}^{\Sigma}\mathrm{IOP}_E$ and implementations over ${}^{\Sigma}\mathrm{IOP}_{E'}$.

The following lemmas relate the conformance relation with LTS languages:

Lemma 4.1
For s' finite LTS over $\mathcal{I}' \times \mathcal{U}'$ s finite LTS over $\mathcal{I} \times \mathcal{U}$, the following holds:
$(L\ s') \subseteq (L\ s)$ *implies* $s' \sqsubseteq_{\mathsf{co}} s$ □

Lemma 4.2
For s' finite LTS over $\mathcal{I}' \times \mathcal{U}'$ s finite LTS over $\mathcal{I} \times \mathcal{U}$ with $\mathcal{I}' \subseteq \mathcal{I}$ the following holds:
$s' \sqsubseteq_{\mathsf{co}} s$ *implies* $(L\ s') \subseteq (L\ s)$ □

The following lemma immediately follows from the above ones:

Lemma 4.3 *For s' finite LTS over $\mathcal{I}' \times \mathcal{U}'$ s finite LTS over $\mathcal{I} \times \mathcal{U}$ with $\mathcal{I}' \subseteq \mathcal{I}$ the following holds:*
$s' \sqsubseteq_{\mathsf{co}} s$ *iff* $(L\ s') \subseteq (L\ s)$ □

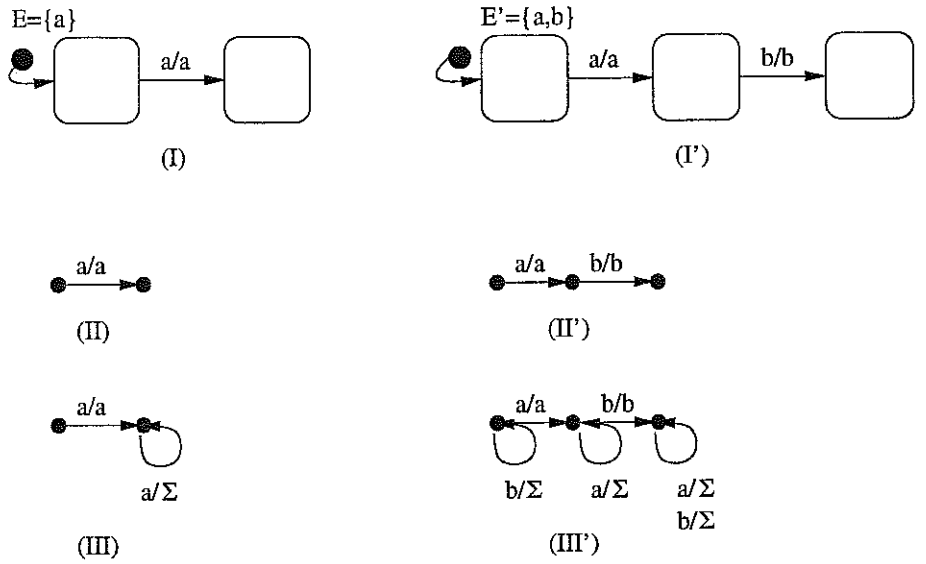

Figure 2: Example 5.1

5 Testing Preorders and the Conformance Relation

Lemma 5.1 *For all HAs* $H = (F, E, \rho), H' = (F', E', \rho')$ *with* $E' \subseteq E$ *the following holds:* $(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathtt{co}} (^{\Sigma}\mathbf{LTS}\ H)$ *implies* $(\mathbf{LTS}\ H') \underset{\sim}{\sqsubseteq}_{MAY} (\mathbf{LTS}\ H)$ □

Lemma 5.2 *For all HAs* $H = (F, E, \rho), H' = (F', E, \rho')$ *the following holds:* $(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathtt{co}} (^{\Sigma}\mathbf{LTS}\ H)$ *implies* $(\mathbf{LTS}\ H) \underset{\sim}{\sqsubseteq}_{MUST} (\mathbf{LTS}\ H')$ □

Notice that in Lemma 5.2 above we require both H and H' have the same input set E. The following examples show that for $H = (F, E, \rho)$ and $H' = (F', E', \rho')$ neither $E \subseteq E'$ alone nor $E' \subseteq E$ alone is enough:

Example 5.1 *Let* $E = \{a\} \subseteq \{a, b\} = E'$, *with* H *(resp.* H'*) as in Fig. 2 (I) (resp. (I')),* $(\mathbf{LTS}\ H)$ *(resp.* $(\mathbf{LTS}\ H')$*) as in Fig. 2 (II) (resp. (II')) and* $(^{\Sigma}\mathbf{LTS}\ H)$ *(resp.* $(^{\Sigma}\mathbf{LTS}\ H')$*) as in Fig. 2 (III) (resp. (III')). Clearly* $(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathtt{co}} (^{\Sigma}\mathbf{LTS}\ H)$, *but* $(\mathbf{LTS}\ H) \not\underset{\sim}{\sqsubseteq}_{MUST} (\mathbf{LTS}\ H')$ *since* $(a, a)(b, b) \in L\ (\mathbf{LTS}\ H') \setminus L\ (\mathbf{LTS}\ H)$ *(see Lemma 3.2).*

Example 5.2 *Let* $E = \{a, b\} \supseteq \{a\} = E'$ *with* H *(resp.* H'*) as in Fig. 3 (I) (resp. (I')),* $(\mathbf{LTS}\ H)$ *(resp.* $(\mathbf{LTS}\ H')$*) as in Fig. 3 (II) (resp. (II')) and* $(^{\Sigma}\mathbf{LTS}\ H)$ *(resp.* $(^{\Sigma}\mathbf{LTS}\ H')$*) as in Fig. 3 (III) (resp. (III')). Clearly* $(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathtt{co}} (^{\Sigma}\mathbf{LTS}\ H)$, *but* $(\mathbf{LTS}\ H) \not\underset{\sim}{\sqsubseteq}_{MUST} (\mathbf{LTS}\ H')$ *since* $\mathcal{AS}\ (\mathbf{LTS}\ H')\ \epsilon = \{\{(a, a)\}\}$ *and* $\mathcal{AS}\ (\mathbf{LTS}\ H)\ \epsilon = \{\{(a, a), (b, b)\}\}$ *which implies* $mfs\ \mathbf{c}\ (\mathcal{AS}\ (\mathbf{LTS}\ H')\ \epsilon) \not\subset\subset mfs\ \mathbf{c}\ (\mathcal{AS}\ (\mathbf{LTS}\ H)\ \epsilon)$.

Notice furthermore that in Lemma 5.2 the implication is strictly one way as shown by the following

Example 5.3 *Let* $E = E' = \{a, b\}$ *with* H *(resp.* H'*) as in Fig. 4 (I) (resp. (I')),* $(\mathbf{LTS}\ H)$ *(resp.* $(\mathbf{LTS}\ H')$*) as in Fig. 4 (II) (resp. (II')) and* $(^{\Sigma}\mathbf{LTS}\ H)$ *(resp.* $(^{\Sigma}\mathbf{LTS}\ H')$*) as in Fig. 4 (III) (resp. (III')). We have* $(\mathbf{LTS}\ H) \underset{\sim}{\sqsubseteq}_{MUST} (\mathbf{LTS}\ H')$ *but* $(^{\Sigma}\mathbf{LTS}\ H') \not\sqsubseteq_{\mathtt{co}} (^{\Sigma}\mathbf{LTS}\ H)$ *since* $\mathtt{OUT}\ (^{\Sigma}\mathbf{LTS}\ H')\ (a, a)(a, \Sigma)\ b = \{b\} \not\subseteq \{\Sigma\} = \mathtt{OUT}\ (^{\Sigma}\mathbf{LTS}\ H)\ (a, a)(a, \Sigma)\ b$.

The following examples show that there is no containment relation between the testing preorder $\underset{\sim}{\sqsubseteq}$ and (the reverse of) the $\sqsubseteq_{\mathtt{co}}$ relation:

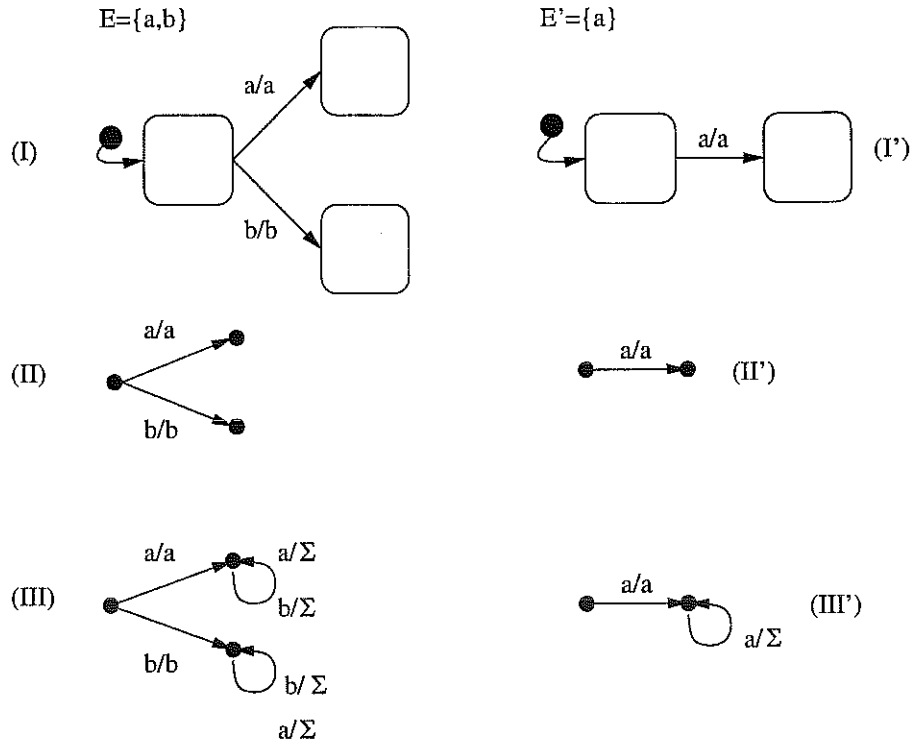

Figure 3: Example 5.2

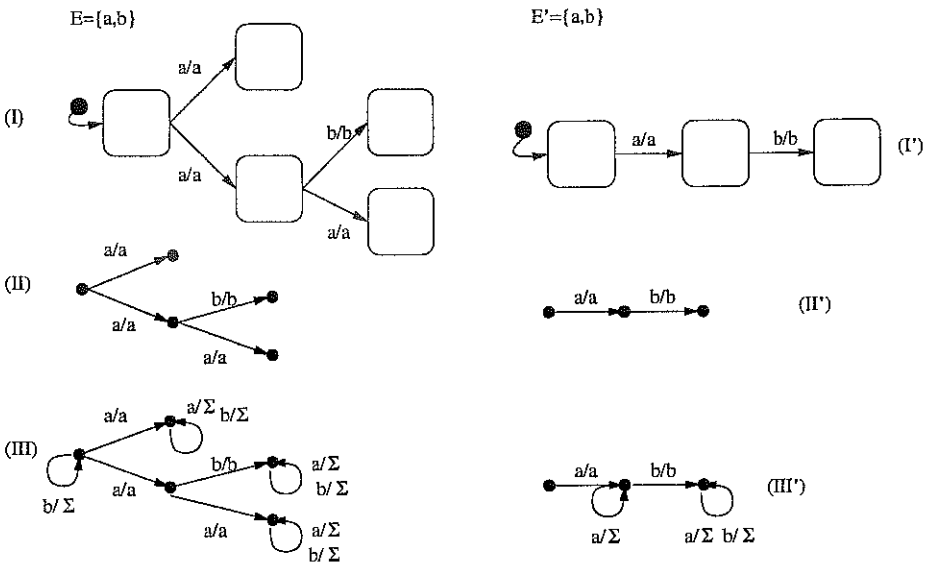

Figure 4: Example 5.3

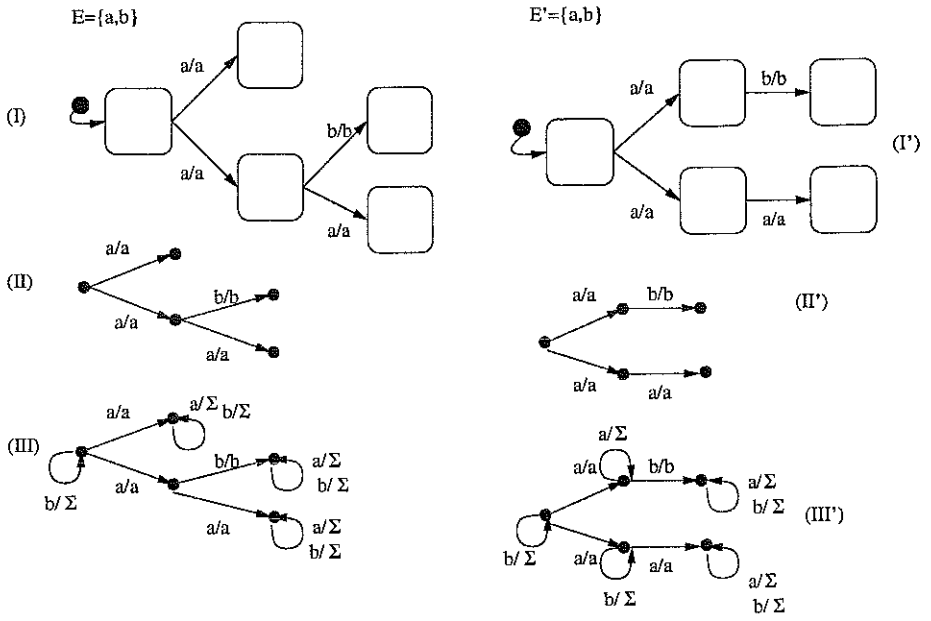

Figure 5: Example 5.4

Example 5.4 *Let* $E = E' = \{a, b\}$ *with* H *(resp.* H'*) as in Fig. 5 (I) (resp. (I')),* (**LTS** H) *(resp.* (**LTS** H')*) as in Fig. 5 (II) (resp. (II')) and* ($^{\Sigma}$**LTS** H) *(resp.* ($^{\Sigma}$**LTS** H')*) as in Fig. 5 (III) (resp. (III')). We have* (**LTS** H) $\sqsubseteq_{\approx}$ (**LTS** H') *since* (**LTS** H) $\sqsubseteq_{\approx_{MAY}}$ (**LTS** H') *(actually* L (**LTS** H) $= L$ (**LTS** H')*) and* (**LTS** H)$\sqsubseteq_{\approx_{MUST}}$(**LTS** H') *but* ($^{\Sigma}$**LTS** H') $\not\sqsubseteq_{\mathtt{co}}$ ($^{\Sigma}$**LTS** H) *since* $\mathtt{OUT}$ ($^{\Sigma}$**LTS** H') $(a,a)(a,\Sigma)$ $b = \{b\} \not\subseteq \{\Sigma\} = \mathtt{OUT}$ ($^{\Sigma}$**LTS** H) $(a,a)(a,\Sigma)$ b.

Example 5.5 *Let* $E = E' = \{a, b\}$ *with* H *(resp.* H'*) as in Fig. 6 (I) (resp. (I')),* (**LTS** H) *(resp.* (**LTS** H')*) as in Fig. 6 (II) (resp. (II')) and* ($^{\Sigma}$**LTS** H) *(resp.* ($^{\Sigma}$**LTS** H')*) as in Fig. 6 (III) (resp. (III')). We have* ($^{\Sigma}$**LTS** H') $\sqsubseteq_{\mathtt{co}}$ ($^{\Sigma}$**LTS** H) *but* (**LTS** H) $\not\sqsubseteq$ (**LTS** H') *since* (**LTS** H) $\not\sqsubseteq_{\approx_{MAY}}$ (**LTS** H') $:(a,a)(b,b) \in L$ (**LTS** H) $\setminus L$ (**LTS** H').

Finally, the following examples show that testing equivalence over the non-stuttering semantics is not strong enough for detecting LTS's capability of refusing to react and, consequently, for discriminating among them on such basis. This in turn implies that testing equivalence does not enjoy substitutivity properties with respect to $\sqsubseteq_{\mathtt{co}}$ (Example 5.7).

Example 5.6 *Let* $E = E' = \{a, b\}$ *with* H *(resp.* H'*) as in Fig. 7 (I) (resp. (I')),* (**LTS** H) *(resp.* (**LTS** H')*) as in Fig. 7 (II) (resp. (II')) and* ($^{\Sigma}$**LTS** H) *(resp.* ($^{\Sigma}$**LTS** H')*) as in Fig. 7 (III) (resp. (III')). We have* (**LTS** H) $\approx$ (**LTS** H') *but* L ($^{\Sigma}$**LTS** H) $\neq L$ ($^{\Sigma}$**LTS** H') *since* $(a,a)(b,\Sigma)(a,a)(b,b) \in L$ ($^{\Sigma}$**LTS** H) $\setminus L$ ($^{\Sigma}$**LTS** H') *and* $(a,a)(b,\Sigma)(a,a)(b,\Sigma) \in L$ ($^{\Sigma}$**LTS** H') $\setminus$ L ($^{\Sigma}$**LTS** H)

Example 5.7 *Take* H *and* H' *as in Example 5.6 and let* $H'' = H$ *From Example 5.6 we know that* (**LTS** H'') $\approx$ (**LTS** H') *and trivially* ($^{\Sigma}$**LTS** H'') $\sqsubseteq_{\mathtt{co}}$ ($^{\Sigma}$**LTS** H).
On the other hand ($^{\Sigma}$**LTS** H') $\not\sqsubseteq_{\mathtt{co}}$ ($^{\Sigma}$**LTS** H) *since* $\mathtt{OUT}$ ($^{\Sigma}$**LTS** H') $(a,a)(b,\Sigma)(a,a)$ $b = \{\Sigma\} \not\subseteq \{b\} = \mathtt{OUT}$ ($^{\Sigma}$**LTS** H) $(a,a)(b,\Sigma)(a,a)$ b. *Similarly, we have that* ($^{\Sigma}$**LTS** H) $\sqsubseteq_{\mathtt{co}}$ ($^{\Sigma}$**LTS** H'') - *trivially - but* ($^{\Sigma}$**LTS** H) $\not\sqsubseteq_{\mathtt{co}}$ ($^{\Sigma}$**LTS** H') *since* $\mathtt{OUT}$ ($^{\Sigma}$**LTS** H) $(a,a)(b,\Sigma)(a,a)$ $b = \{b\} \not\subseteq \{\Sigma\} =$ $\mathtt{OUT}$ ($^{\Sigma}$**LTS** H') $(a,a)(b,\Sigma)(a,a)$ b.

The above examples show that (testing equivalence based on) the non-stuttering semantics is not adequate for conformance testing in the sense that one cannot replace (testing) equivalent LTSs

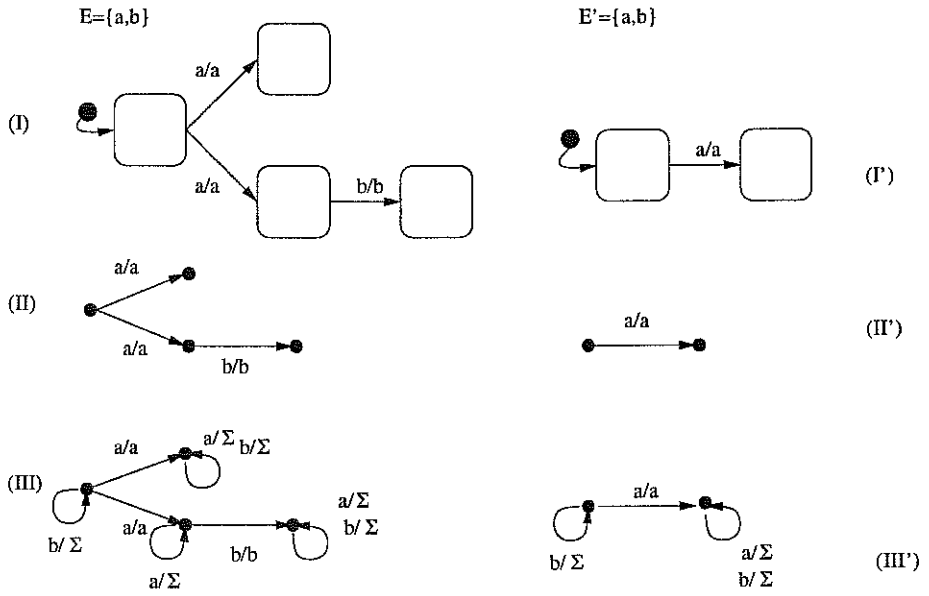

Figure 6: Example 5.5

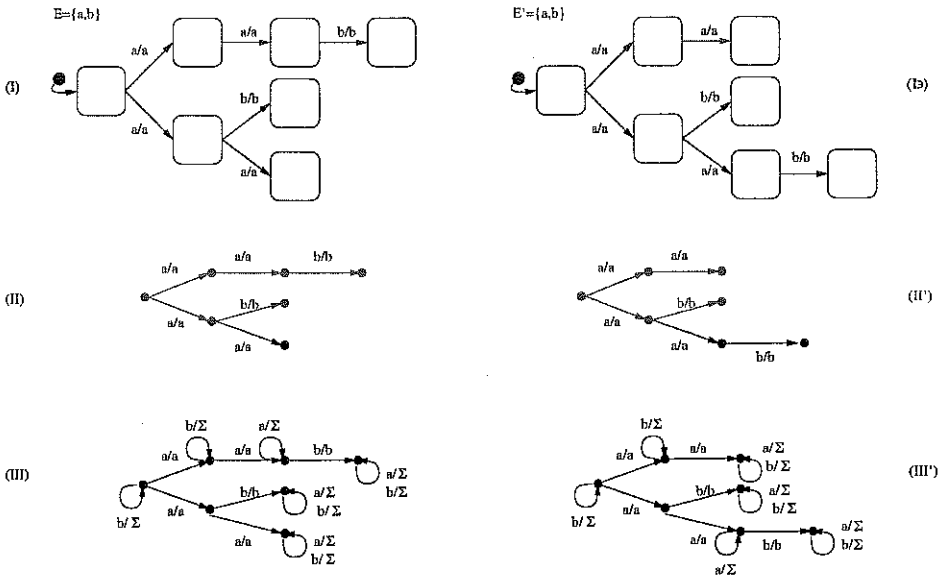

Figure 7: Example 5.6

still preserving conformance. More specifically equivalent implementations are not conformant with the same specification. Similarly, the same implementation turns out not to be conformant to equivalent specifications. Such inadequacy comes from the fact that (the experimenters testing those LTSs generated according to) the non-stuttering semantics are unable detect absence of reaction and to take proper actions when this happens. Due to the non-stuttering semantics, experimental systems can only deadlock in such situations.

In order to be adequate, the semantics must explicitly deal with stuttering, so that experimenters can detect absence of reaction and behave accordingly. This intuitive consideration is supported by Lemmas 5.3 and 5.4 below:

Lemma 5.3 *For all HAs* $H = (F, E, \rho), H' = (F', E', \rho')$ *the following holds:*

i) $(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\sim_{MAY}} (^{\Sigma}\mathbf{LTS}\ H')$ *implies* $(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H')$

ii) $(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H')$ *and* $E \subseteq E'$ *implies* $(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\sim_{MAY}} (^{\Sigma}\mathbf{LTS}\ H')$

□

Lemma 5.4 *For all HAs* $H = (F, E, \rho), H' = (F', E', \rho')$ *the following holds:*
$(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\sim_{MUST}} (^{\Sigma}\mathbf{LTS}\ H')$ *implies* $(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H)$ □

The following theorem establishes the adequacy of the testing relations based on the stuttering semantics for the conformance relation:

Theorem 5.1 *For all HAs* $H = (F, E, \rho), H' = (F', E', \rho')$ *and* $H'' = (F'', E'', \rho'')$ *with* $E' \subseteq E$ *the following holds:*

i) $(^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\sim_{MAY}} (^{\Sigma}\mathbf{LTS}\ H') \wedge (^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H)$ *implies*
$(^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H)$

ii) $(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\sim_{MUST}} (^{\Sigma}\mathbf{LTS}\ H'') \wedge (^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H)$ *implies*
$(^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H)$

iii) $(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\simeq} (^{\Sigma}\mathbf{LTS}\ H'') \wedge (^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H)$ *implies*
$(^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H)$

iv) $(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H) \wedge (^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\sim_{MAY}} (^{\Sigma}\mathbf{LTS}\ H'')$ *implies*
$(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H'')$

v) $(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H) \wedge (^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\sim_{MUST}} (^{\Sigma}\mathbf{LTS}\ H)$ *implies*
$(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H'')$

vi) $(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H) \wedge (^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\simeq} (^{\Sigma}\mathbf{LTS}\ H)$ *implies*
$(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H'')$

□

Corollary 5.1 *For all HAs* $H = (F, E, \rho), H' = (F', E', \rho')$ *and* $H'' = (F'', E'', \rho'')$ *with* $E' \subseteq E$ *the following holds:*

i) $(^{\Sigma}\mathbf{LTS}\ H') \simeq (^{\Sigma}\mathbf{LTS}\ H'') \wedge (^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H)$ *implies*
$(^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H)$

ii) $(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H) \wedge (^{\Sigma}\mathbf{LTS}\ H'') \simeq (^{\Sigma}\mathbf{LTS}\ H)$ *implies*
$(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathrm{co}} (^{\Sigma}\mathbf{LTS}\ H'')$

We conclude this section with the following three lemmas linking the non-stuttering semantics and the stuttering one via the testing relations in the way one would expect:

Lemma 5.5 *For all HAs* $H = (F, E, \rho), H' = (F', E', \rho')$ *the following holds:*
$(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\sim MAY} (^{\Sigma}\mathbf{LTS}\ H')$ *implies* $(\mathbf{LTS}\ H) \sqsubseteq_{\sim MAY} (\mathbf{LTS}\ H')$ □

Lemma 5.6 *For all HAs* $H = (F, E, \rho), H' = (F', E, \rho')$ *the following holds:*
$(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\sim MUST} (^{\Sigma}\mathbf{LTS}\ H')$ *implies* $(\mathbf{LTS}\ H) \sqsubseteq_{\sim MUST} (\mathbf{LTS}\ H')$ □

The following lemma directly follows from Lemmas 5.5 and 5.6 above:

Lemma 5.7 *For all HAs* $H = (F, E, \rho), H' = (F', E, \rho')$ *the following holds:*
$(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\sim} (^{\Sigma}\mathbf{LTS}\ H')$ *implies* $(\mathbf{LTS}\ H) \sqsubseteq_{\sim} (\mathbf{LTS}\ H')$ □

6 Conclusions

In this paper we investigated the formal relationship between testing preorder/equivalences for a behavioural subset of UML Statechart Diagrams (UMLSDs) and a conformance relation for implementations with respect to specifications given using such diagrams.

We showed that the conformance relation for UMLSDs proposed in [5] is strictly stronger than the reverse of the MUST preorder based on the non-stuttering semantics, as introduced in [11], and so stronger that the companion MAY preorder.

Moreover, the Testing preorder and the conformance relation are incomparable; neither one is stronger than the other nor vice-versa. Furthermore, no substitutivity property holds: replacing an implementation conforming to a specification with a testing equivalent implementation may brake conformance; symmetrically, an implementation conforming to a specification is not guaranteed to conform also to another, testing equivalent, specification.

On the basis of the above negative results, we adopted a *stuttering* semantics also for the general testing theory. This amounts to giving experimenters the power of recognizing absence of system reaction and behaving accordingly.

We showed that, in this case, the MAY preorder is stronger that $\sqsubseteq_{co}$, which implies that $\sqsubseteq_{co}$ contains the reverse of the MUST preorder. Thus the Testing preorder is stronger than the reverse conformance relation. As a consequence, one can replace testing equivalent specifications and implementations still preserving their conformance relation.

7 Acknowledgements

The authors would like to thank Jan Tretmans and Ina Schieferdecker for the fruitful discussions and suggestions which helped them developing the work discussed in the present paper.

References

[1] B. Bosik and M. Ümit Uyar. Finite state machines based formal methods in protocol conformance testing: from theory to implementation. *Computer Networks and ISDN Systems. North-Holland*, 22:7–33, 1991.

[2] J. Broersen and R. Wieringa. Interpreting UML-statecharts in a modal μ-calculus. Unpublished manuscript, 1997.

[3] S. Gnesi, D. Latella, and M. Massink. Modular semantics for a UML Statechart Diagrams kernel and its extension to Multicharts and Branching Time Model Checking. *The Journal of Logic and Algebraic Programming. Elsevier Science.* (To appear).

[4] S. Gnesi, D. Latella, and M. Massink. Model checking UML statechart diagrams using JACK. In A. Williams, editor, *Fourth IEEE International High-Assurance Systems Engineering Symposium*, pages 46–55. IEEE Computer Society Press, 1999. ISBN 0-7695-0418-3.

[5] S. Gnesi, D. Latella, and M. Massink. Formal conformance testing UML Statechart Diagrams Behaviours: From theory to automatic test generation. Technical Report CNUCE-B04-2001-16, Consiglio Nazionale delle Ricerche, Istituto CNUCE, 2001. (Full version).

[6] M. Hennessy. *Algebraic Theory of Processes.* The MIT Press, 1988.

[7] D. Latella, I. Majzik, and M. Massink. A simplified formal operational semantics for a subset of UML statechart diagrams. Technical Report HIDE/T1.2/PDCC/5/v1, ESPRIT Project n. 27439 - High-Level Integrated Design Environment for Dependability HIDE, 1998.

[8] D. Latella, I. Majzik, and M. Massink. Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing. The International Journal of Formal Methods. Springer*, 11(6):637–664, 1999.

[9] D. Latella, I. Majzik, and M. Massink. Towards a formal operational semantics of UML statechart diagrams. In P. Ciancarini, A. Fantechi, and R. Gorrieri, editors, *IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Oriented Distributed Systems*, pages 331–347. Kluwer Academic Publishers, 1999. ISBN 0-7923-8429-6.

[10] D. Latella and M. Massink. A formal testing framework for UML Statechart Diagrams Behaviours: From theory to automatic verification. Technical Report CNUCE-B4-2001-007, Consiglio Nazionale delle Ricerche, Istituto CNUCE, 2001. (Full version).

[11] D. Latella and M. Massink. A formal testing framework for UML Statechart Diagrams behaviours: From theory to automatic verification. In *Sixth IEEE International High-Assurance Systems Engineering Symposium.* IEEE Computer Society Press, 2001. (To appear).

[12] D. Lee and M. Yannakakis. Principles and Methods of Testing Finite State Machines - A Survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[13] J. Lilius and I. Paltor Porres. Formalising UML state machines for model checking. In R. France and B. Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard.*, volume 1723 of *Lecture Notes in Computer Science*, pages 430–445. Springer-Verlag, 1999.

[14] J. Lilius and I. Paltor Porres. The semantics of UML state machines. Technical Report 273, Turku Centre for Computer Science, 1999.

[15] Z. Manna, S. Ness, and J. Vuillemin. Inductive methods for proving properties of programs. *Communications of the ACM*, 16(8):491–502, 1973.

[16] M. Massink. *Functional Techniques in Concurrency.* PhD thesis, University of Nijmegen, February 1996. ISBN 90-9008940-3.

[17] R. Milner. *Communication and Concurrency.* Series in Computer Science. Prentice Hall, 1989.

[18] Object Management Group, Inc. *OMG Unified Modeling Language Specification - version 1.3*, 1999.

[19] Rational Software and Microsoft and Hewlett-Packard and Oracle and Sterling Software and MCI Systemhouse and Unisys and ICON Computing and IntelliCorp and i-Logix and IBM and ObjecTime and Platinum Technology and Ptech and Taskon and Reich Technologies and Softeam. *UML Semantics, version 1.1*, 1997. UML semantics with metamodel.

[20] J. Tretmans. *A Formal Approach to Conformance Testing.* PhD thesis, University of Twente, 1992.

[21] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 17(3):103–120, 1996.

[22] J. Tretmans. Testing concurrent systems: A formal approach. In J. Baeten and S. Mauw, editors, *Concur '99*, volume 1664 of *Lecture Notes in Computer Science*, pages 46–65. Springer-Verlag, 1999.

[23] R. Wieringa and J. Broersen. A minimal transition system semantics for lightweight class and behavior diagrams. In M. Broy, D. Coleman, T. Maibaum, and B. Rumpe, editors, *Proceedings of the ICSE98 Workshop on Precise Semantics for Software Modeling techniques*, 1998.

APPENDIX

A The core semantics

In this section we recall the main definitions related to the core semantics. For a complete treatment of its features the reader is referred to [9, 4].

In the context of HAs, the transition labels of sequential automaton A have a particular structure. For transition t we require its label to be a 5-tuple (sr, ev, g, ac, td), where sr is the *source restriction*, ev is the *trigger* event, g is the *guard*, ac is the *actions* list and td is the *target determinator*. In the sequel we use the following functions SRC, TGT, SR, EV, G, AC, TD, defined in the obvious way: for transition $t = (s, (sr, ev, g, ac, td), s')$, $SRC\ t = s, TGT\ t = s', SR\ t = sr, EV\ t = ev, G\ t = g, AC\ t = ac, TD\ t = td$. $EV\ t$ is the event which triggers transition t, $G\ t$ is a boolean guard which must evaluate to true in order for t to be enabled, while $AC\ t$ are the actions which result from firing t. $SR\ t$ and $TD\ t$ are intimately related to the way we represent interlevel transitions. For details the reader is referred to [9].

Given HA $H = (F, \rho, E)$, every sequential automaton $A \in F$ characterises a HA in its turn: intuitively, such a HA is composed by all those sequential automata which lay below A, including A itself, and has a refinement function ρ_A which is a proper restriction of ρ. A is the root automaton.

Definition A.1
For $A \in F$ the automata, states, labels *and* transitions under A *are defined respectively as*
$\mathcal{A}\ A = \{A\} \cup (\bigcup_{A' \in \left(\bigcup_{s \in \sigma_A} (\rho_A\, s)\right)} (\mathcal{A}\ A'))$,
$\mathcal{S}\ A = \bigcup_{A' \in \mathcal{A}\ A} \sigma_{A'}$,
$\mathcal{T}\ A = \bigcup_{A' \in \mathcal{A}\ A} \delta_{A'}$. ∘

The definition of sub-HA follows:

Definition A.2 (Sub Hierarchical Automata)
For $A \in F$, (F_A, E, ρ_A), where $F_A = (\mathcal{A}\ A)$, and $\rho_A = \rho_{|(\mathcal{S}\ A)}$, is the HA characterised by A. ∘

In the sequel for $A \in F$ we shall refer to A both as a sequential automaton and as the sub-HA of H it characterises, the role being clear from the context. H will be identified with A_{root}. Sequential Automata will be considered a degenerate case of HAs.

In [9] the notions of *conflicting transitions*, *state precedence*, *transition priority* and *orthogonal states* are formally defined. The interested reader is referred to the above mentioned paper. When transitions t and t' are in conflict we write $t \# t'$. Priorities are assigned to transitions via a function π and a partial order $\sqsubseteq$ based on state precedence is defined on them. So, we say that t has lower priority than (the same priority as) t' iff $\pi t \sqsubseteq \pi t'$. The transition firing relation $\xrightarrow{L}$ is defined as the smallest relation over $F \times 2^{\mathcal{T}\ H} \times \mathrm{Conf}_H \times \Theta_E \times 2^{\mathcal{T}\ H} \times \mathrm{Conf}_H \times \Theta_E$ induced by the deduction system defined in Fig. 8, where the following auxiliary functions are used:

Definition A.3 (Enabled Transitions)
For $A \in F$, set of states $\mathcal{C}$ and environment $\mathcal{E}$,

(i) the set of all the enabled local *transitions of A in $(\mathcal{C}, \mathcal{E})$, $LE_A\ \mathcal{C}\ \mathcal{E}$ is defined as follows*[3]*:*

$$LE_A\ \mathcal{C}\ \mathcal{E} = \{t \in \delta_A \mid \{(SRC\ t)\} \cup (SR\ t) \subseteq \mathcal{C}\ \wedge (EV\ t) \in \mathcal{E} \wedge (\mathcal{C}, \mathcal{E}) \models (G\ t)\}$$

(ii) the set of all enabled *transitions of A in $(\mathcal{C}, \mathcal{E})$ considered as an HA, i.e. including those of descendents of A, $E_A\ \mathcal{C}\ \mathcal{E}$ is defined as follows:*

[3] $(\mathcal{C}, \mathcal{E}) \models g$ means that guard g is true of $(\mathcal{C}, \mathcal{E})$. Its formalisation is immaterial for the purposes of the present paper. In the deduction rules, we will relax the requirement $\mathcal{C} \in \mathrm{Conf}_A$ and we will assume $\mathcal{C} \in \mathrm{Conf}_H$. This allows the use of guards which make reference to non local states.

Progress Rule

$$\frac{\begin{array}{ll} t \in LE_A\ \mathcal{C}\ \mathcal{E} & (1) \\ \nexists t' \in P \cup E_A\ \mathcal{C}\ \mathcal{E}.\ \pi\ t \sqsubset \pi\ t' & (2) \end{array}}{A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\{t\}} (DEST\ t, new(ACt))}$$

Composition Rule

$$\frac{\begin{array}{ll} \{s\} = \mathcal{C} \cap \sigma_A & (1) \\ \rho_A\ s = \{A_1, \ldots, A_n\} \neq \emptyset & (2) \\ \left(\bigwedge_{j=1}^{n} A_j \uparrow (P \cup LE_A\ \mathcal{C}\ \mathcal{E}) :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L_j} (\mathcal{C}_j, \mathcal{E}_j)\right) \wedge is\_join_{j=1}^{n} \mathcal{E}_j\ \mathcal{J} & (3) \\ \left(\bigcup_{j=1}^{n} L_j = \emptyset\right) \Rightarrow (\forall t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P.\ \pi t \sqsubset \pi t') & (4) \end{array}}{A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\bigcup_{j=1}^{n} L_j} (\{s\} \cup \bigcup_{j=1}^{n} \mathcal{C}_j, \mathcal{J})}$$

Stuttering Rule

$$\frac{\begin{array}{ll} \{s\} = \mathcal{C} \cap \sigma_A & (1) \\ \rho_A\ s = \emptyset & (2) \\ \forall t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P.\ \pi t \sqsubset \pi t' & (3) \end{array}}{A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\emptyset} (\{s\}, nil)}$$

Figure 8: Operational Semantics of UML Hierarchical Automata

$$E_A\ \mathcal{C}\ \mathcal{E} = \bigcup_{A' \in (\mathcal{A}\ A)} LE_{A'}\ \mathcal{C}\ \mathcal{E}$$

Finally, for transition t, $(DEST\ t)$ denotes the set $\{s \mid \exists s' \in (TD\ t).\ (TGT\ t) \preceq s \preceq s'\}$.
As usual, in the following we will often write $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L}$ as a shorthand for $\exists \mathcal{C}', \mathcal{E}'.\ A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$

Many of the proofs we give in this section are carried out by induction either on the length d of the derivation for proving $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$ [17] or on the structure of the subset of F affected by $\mathcal{C}$. With respect to structural induction, let $F_{\mathcal{C}}$ be the set $\{A \in F \mid \mathcal{C} \cap \sigma_A \neq \emptyset\}$. It is easy to define a relation on $F_{\mathcal{C}}$ such that X is related Y iff $s \in \mathcal{C} \cap \sigma_Y$ and $X \in (\rho_Y s)$. Notice that since $\mathcal{C}$ is a configuration, for each A in $F_{\mathcal{C}}$ there is a unique state $s \in \sigma_A \cap \mathcal{C}$. The transitive and reflexive closure of such a relation is a well-founded partial order, since antisymmetry is a consequence of property (iii) in the definition of HAs; the bottom elements are those X such that $\rho\sigma_X = \emptyset$ [15].

The following lemma gives some insights on the core semantics and will be used in the following sections:

Lemma A.1
For all HA $H = (F, E, \rho), A \in F, P \subseteq (\mathcal{T}\ H), \mathcal{E} \in \Theta_E, \mathcal{C} \in \mathrm{Conf}_H$, *s.t.* $\sigma_A \cap \mathcal{C} \neq \emptyset$ *the following holds:*

i) $\exists L \subseteq (\mathcal{T}\ H), \mathcal{C}' \in \mathrm{Conf}_H, \mathcal{E}' \in \Theta_E.\ A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$

ii) $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\emptyset} (\mathcal{C}', \mathcal{E}')$ *and* $\mathcal{C} \in \mathrm{Conf}_A$ *implies* $\mathcal{C}' = \mathcal{C}$ *and* $\mathcal{E}' = nil$

iii) $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\emptyset} (\mathcal{C}, nil)$ *implies* $\nexists L \neq \emptyset, \mathcal{C}' \in \mathrm{Conf}_H, \mathcal{E}' \in \Theta_E.\ A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$

Proof

Part i -
By induction on the structure of $F_{\mathcal{C}}$.

Base case ($\rho_A\ s = \emptyset$ where $\{s\} = \mathcal{C} \cap \sigma_A$):
If $\forall t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P.\ \pi t \sqsubset \pi t'$ then the Stuttering Rule can be applied immediately to prove the assert. If instead $\exists t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \nexists t' \in P.\ \pi\ t \sqsubset \pi\ t'$ then the Progress Rulecan be applied immediately to prove the assert.

Induction step ($\rho_A\ s = \{A_1, \ldots, A_n\}$where $\{s\} = \mathcal{C} \cap \sigma_A$):
If $\exists t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \nexists t' \in P \cup E_A\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t'$ then the Progress Rule can be applied immediately to prove the assert. Suppose then that $\forall t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P \cup E_A\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t'$. By the Induction Hypothesis we know that $A_j \uparrow P \cup (LE_A\ \mathcal{C}\ \mathcal{E}) :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L_j} (\mathcal{C}_j, \mathcal{E}_j)$ for $j = 1, \ldots, n$ where $\rho_A s = \{A_1, \ldots, A_n\}$ with $\{s\} = \mathcal{C} \cap \sigma_A$, so that we can apply the Composition Rule for getting the assert. To that purpose we must be sure that also premise (4) of the Composition Rule is fulfilled, which we show in the sequel. From the hypothesis $\forall t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P \cup E_A\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t'$, using set theory, Lemma A.2 below, and noting that δ_A is finite, we can first of all conclude

$$\forall t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P \cup \left(\bigcup_{j=1\ldots n} E_{A_j}\ \mathcal{C}\ \mathcal{E}\right).\ \pi t \sqsubset \pi t' \quad \text{(I)}$$

Moreover, if $\bigcup_{j=1\ldots n} L_j = \emptyset$, by Lemma A.3 below we get also

$$\forall t \in LE_{A_j}\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P \cup LE_A\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t' \quad \text{(II)}$$

Since all sets involved are finite, by combining (I) and (II) above together we get, from set theory, premise (4) of the Composition Rule.

Part ii -
By derivation induction.

Base case ($d = 1$):
In this case only the Stuttering Rule could have been applied. Moreover $\mathcal{C} \in \mathrm{Conf}_A$ and $\{s\} = \sigma_A \cap \mathcal{C}$ implies $\mathcal{C} = \{s\}$.

Induction step ($d > 1$):
In this case the Composition Rule must have been applied in the last step of the derivation.
$L = \emptyset$

$\Rightarrow$ {Def. of Composition Rule}

$\bigwedge_{j=1,\ldots,n} A_j \uparrow P \cup LE_A\ \mathcal{C}\ \mathcal{E} :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\emptyset} (\mathcal{C}'_j, \mathcal{E}'_j)$

$\Rightarrow$ {$\mathcal{C} \in \mathrm{Conf}_A, s \in \mathcal{C} \cap \sigma_A$, Def. 2.3 implies $\mathcal{C} = \{s\} \cup \left(\bigcup_{j=1,\ldots,n} \mathcal{C}_j\right)$ with $\mathcal{C}_j \in \mathrm{Conf}_{A_j}$; Lemma A.4 below}

$\bigwedge_{j=1,\ldots,n} A_j \uparrow P \cup LE_A\ \mathcal{C}\ \mathcal{E} :: (\mathcal{C}_j, \mathcal{E}) \xrightarrow{\emptyset} (\mathcal{C}''_j, \mathcal{E}''_j)$

$\Rightarrow$ {Induction Hypothesis}

$\bigwedge_{j=1,\ldots,n} \mathcal{C}_j = \mathcal{C}''_j \wedge \mathcal{E}''_j = nil$

$\Rightarrow$ {Def. of Composition Rule}

$\mathcal{C}' = \mathcal{C} \wedge \mathcal{E}' = nil$

Part iii -

By derivation induction.

Base case ($d = 1$):
In this case only the Stuttering Rule could have been applied, which trivially proves the assert.

Induction step ($d > 1$):
In this case only the Composition Rule could have been applied in the last step of the derivation. In this case, $L = \emptyset$ implies $A_j \uparrow (P \cup LE_A\ \mathcal{C}\ \mathcal{E}) :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\emptyset} (\mathcal{C}_j, \mathcal{E}_j)$ for $j = 1, \ldots, n$, which, for the Induction Hypothesis brings to $/\exists L_j \neq \emptyset.\ A_j \uparrow (P \cup LE_A\ \mathcal{C}\ \mathcal{E}) :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L_j}$ for $j = 1, \ldots, n$. Thus the Composition Rule cannot be applied in any other way for producing a transition $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$ with $L \neq \emptyset$. □

In the above proof we used the following lemmas:

Lemma A.2
For all HA $H = (F, E, \rho), A \in F, \mathcal{E} \in \Theta_E, \mathcal{C} \in \mathrm{Conf}_H$ *where* $\sigma_A \cap \mathcal{C} = \{s\}$ *and* $\rho_A s = \{A_1, \ldots, A_n\}$ *the following holds:* $E_A\ \mathcal{C}\ \mathcal{E} = LE_A\ \mathcal{C}\ \mathcal{E} \cup \left(\bigcup_{j=1,\ldots,n} E_{A_j}\ \mathcal{C}\ \mathcal{E}\right)$

Proof

By induction on the structure of $F_{\mathcal{C}}$.

Base case ($\rho_A\ s = \emptyset$ where $\{s\} = \mathcal{C} \cap \sigma_A$):
Trivially $E_A\ \mathcal{C}\ \mathcal{E} = LE_A\ \mathcal{C}\ \mathcal{E}$ if $\rho_A\ s = \emptyset$

Induction step ($\rho_A\ s = \{A_1, \ldots, A_n\}$where $\{s\} = \mathcal{C} \cap \sigma_A$):

$E_A\ \mathcal{C}\ \mathcal{E}$

$=$ {Def. A.3 (E_A)}

$\bigcup_{A' \in (\mathcal{A}\ A)} LE_{A'}\ \mathcal{C}\ \mathcal{E}$

$=$ {Def. A.1 ($\mathcal{A}$)}

$\bigcup_{A' \in \{A\} \cup \left(\bigcup_{A'' \in (\rho_A \sigma_A)} \mathcal{A}\ A''\right)} LE_{A'}\ \mathcal{C}\ \mathcal{E}$

$=$ {Def. of σ_A and $\rho_A s$}

$\bigcup_{A' \in \{A\} \cup \left(\bigcup_{j=1,\ldots,n} \mathcal{A}\ A_j\right) \cup \left(\bigcup_{A'' \in (\rho_A \{s' \in \sigma_A \mid s' \neq s\})} \mathcal{A}\ A''\right)} LE_{A'}\ \mathcal{C}\ \mathcal{E}$

$=$ {$s' \in \sigma_A$ with $s' \neq s$ and $A'' \in \rho_a s'$ implies $LE_{A''}\ \mathcal{C}\ \mathcal{E} = \emptyset$}

$\bigcup_{A' \in \{A\} \cup \left(\bigcup_{j=1,\ldots,n} \mathcal{A}\ A_j\right)} LE_{A'}\ \mathcal{C}\ \mathcal{E}$

$=$ {Set Theory}

$LE_A\ \mathcal{C}\ \mathcal{E} \cup \left(\bigcup_{j=1,\ldots,n} \bigcup_{A' \in \mathcal{A}\ A_j} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right)$

$=$ {Def. A.3 (E_A)}

$LE_A\ \mathcal{C}\ \mathcal{E} \cup \left(\bigcup_{j=1,\ldots,n} E_{A_j}\ \mathcal{C}\ \mathcal{E}\right)$ □

Lemma A.3
For all HA $H = (F, E, \rho), A \in F, P \subseteq \mathcal{T}\ H, \mathcal{E} \in \Theta_E, \mathcal{C} \in \mathrm{Conf}_H$ *the following holds:*
$A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\emptyset}$ *implies* $\forall t \in E_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P.\ \pi t \sqsubset \pi t'$

Proof

By derivation induction.

Base case ($d = 1$):
In this case only the Stuttering Rule could have been applied and the assert follows trivially.

Induction step ($d > 1$):
In this case the Composition Rule must have been applied in the last step of the derivation. Since $\bigcup_{j=1\ldots n} L_j = \emptyset$, we know

$$\forall t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P.\ \pi t \sqsubset \pi t' \quad \text{(I)}$$

by premise (4) of the Composition Rule, and

$$\forall t \in E_{A_j}\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P \cup LE_A\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t' \quad \text{(II)}$$

for $j = 1, \ldots, n$ because of the Induction Hypothesis. By using Lemma A.2 and I and II above we get the assert. □

Lemma A.4
For all HA $H = (F, E, \rho), A \in F, P \subseteq (\mathcal{T}\ H), \mathcal{E} \in \Theta_E, \mathcal{C} \in \mathrm{Conf}_H$, *s.t.* $\sigma_A \cap \mathcal{C} \neq \emptyset$, *the following holds: if* $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\emptyset}$, $\mathcal{C}' \in \mathrm{Conf}_A$ *and* $\mathcal{C}' \subseteq \mathcal{C}$ *then* $A \uparrow P :: (\mathcal{C}', \mathcal{E}) \xrightarrow{\emptyset}$

Proof

We proceed derivation induction.
Base case ($d = 1$):
If the derivation has length 1, then only the Stuttering Rule could have been applied. Moreover $\mathcal{C}' \in \mathrm{Conf}_A$, $\mathcal{C}' \subseteq \mathcal{C}$ and $\{s\} = \sigma_A \cap \mathcal{C}$ implies, in this case, $\mathcal{C}' = \{s\}$. Thus the assert follows.
Induction step ($d > 1$):
In this case the Composition Rule must have been applied in the last step of the derivation, which, together with $\mathcal{C}' \in \mathrm{Conf}_A$ and $\mathcal{C}' \subseteq \mathcal{C}$, implies $\mathcal{C}' = \{s\} \cup \left(\bigcup_{j=1,\ldots,n} \mathcal{C}'_j\right)$ where $\mathcal{C}'_j \in \mathrm{Conf}_{A_j}$ for $j = 1, \ldots, n$. We know, $\bigwedge_{j=1,\ldots,n} A_j \uparrow P \cup LE_A\ \mathcal{C}\ \mathcal{E} :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\emptyset}$ and obviously $\mathcal{C}'_j \subseteq \mathcal{C}$. So, by Induction Hypothesis, we get $A_j \uparrow P \cup LE_A\ \mathcal{C}\ \mathcal{E} :: (\mathcal{C}'_j, \mathcal{E}) \xrightarrow{\emptyset}$. Moreover, we note that $\forall t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P.\ \pi t \sqsubset \pi t' \wedge \mathcal{C}' \subseteq \mathcal{C}$ implies $\forall t \in LE_A\ \mathcal{C}'\ \mathcal{E}.\ \exists t' \in P.\ \pi t \sqsubset \pi t'$. Thus, by applying the Composition Rule we get $A \uparrow P :: (\mathcal{C}', \mathcal{E}) \xrightarrow{\emptyset}$ that is, the assert. □

B Environments notation

In [10] a notation for specifying environment behaviours has been introduced which is a (rather free) mix of process algebra (guarded action prefix, choice, and process definition/instantiation) and lambda-calculus, plus the use of our notation for abstract data types. Fig. 9 gives an account of the operational semantics of the notation we use, as far as the process algebra component is concerned; action prefix is denoted by '$e; \mathcal{I}$' - or $\alpha; \mathcal{O}$ - with $\alpha \in$ SPECIAL, guarding is denoted by '$g \Rightarrow \mathcal{O}$', choice is denoted by $+$, process definition (resp. instantiation) is denoted by $P(\mathcal{F} : \mathcal{T}) := \mathcal{O}$ (resp. $P(\mathcal{E})$). In Fig. 9 $\mu \in E \cup$SPECIAL. For the functional part we just refer to standard lambda-calculus and to our previous discussion on abstract data types, only pointing out that guard evaluation may involve proper binding of variables; the notation $[\mathcal{E}/\mathcal{F}]$ denotes (data) variable substitution. A formal detailed definition of such a notation is out of the scope of the present paper.

$$e;\mathcal{I} \xrightarrow{e} \mathcal{I} \qquad \alpha;\mathcal{O} \xrightarrow{\alpha} \mathcal{O} \qquad \frac{g \equiv \text{TRUE}, \mathcal{O} \xrightarrow{e} \mathcal{I}}{g \Rightarrow \mathcal{O} \xrightarrow{e} \mathcal{I}}$$

$$\frac{\mathcal{O} \xrightarrow{\mu} T}{\mathcal{O} + \mathcal{O}' \xrightarrow{\mu} T} \qquad \frac{\mathcal{O} \xrightarrow{\mu} T}{\mathcal{O}' + \mathcal{O} \xrightarrow{\mu} T} \qquad \frac{P(\mathcal{F} : T) := \mathcal{O}, \mathcal{O}[\mathcal{E}/\mathcal{F}] \xrightarrow{e} \mathcal{I}}{P(\mathcal{E}) \xrightarrow{e} \mathcal{I}}$$

Figure 9: Environment Operational Semantics

C The original semantics

In this section we recall the definition of the operational semantics given in [4, 3]. In [3] such semantics has been proven consistent with major behavioural requirements informally defined in [19, 18].

Definition C.1 *The operational semantics of an HA H is the LTS $(\mathbf{S}, \mathbf{s^0} Act, \longrightarrow)$ where (i) $\mathbf{S} = \text{Conf}_H \times (\Theta\ E)$ is the set of statuses of LTS, (ii) $\mathbf{s^0} = (\mathcal{C}_0, \mathcal{E}_0) \in \mathbf{S}$ is the initial status, (iii)$Act \subseteq 2^{\mathcal{T}\ H}$ is the set of actions of the LTS and (iv) $\longrightarrow \subseteq \mathbf{S} \times Act \times \mathbf{S}$ is the smallest relation induced by the following rule:*

$$\frac{(Sel\ \mathcal{E}\ e\ \mathcal{E}'') \quad H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')}{(\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', (join\ \mathcal{E}''\ \mathcal{E}'))}$$

Notice that the operational semantics definition is based on the *same* core semantics as that of the present paper. In [7, 3] we proved the following theorem, which states that our semantics fulfills major behavioural features required by the official UML definition:

Theorem C.1 *For all $L \subseteq (\mathcal{T}\ A)$, $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L}$ if and only if L is a maximal set, under set inclusion, which satisfies all the following properties: (i) L is conflict-free, i.e. $\forall t, t' \in L.\ \neg t \# t'$; (ii) all transitions in L are enabled in the current status, i.e. $L \subseteq E_A\ \mathcal{C}\ \mathcal{E}$; (iii) there is no transition outside L which is enabled in the current status and which has higher priority than a transition in L, i.e. $\forall t \in L.\ \nexists t' \in E_A\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t'$; and (iv) all transitions in L respect P, i.e. $\forall t \in L.\ \nexists t' \in P.\ \pi t \sqsubset \pi t'$*

D Detailed Proofs

D.1 Proof of Lemma 2.1

$u \in (\texttt{OUT}\ s\ \gamma\ i)$

$\equiv$ {Def. of OUT}

$\exists s'.\ (s \xrightarrow{\gamma} s') \wedge s' \xrightarrow{(i,u)}$

$\equiv$ {Def. of $\xrightarrow{\gamma(i,u)}$}

$\exists s'.\ s \xrightarrow{\gamma(i,u)} s'$

$\equiv$ {Def. of L}

$\gamma(i,u) \in (L\ s)$ □

D.2 Proof of Theorem 2.3

Part i -

$C \xrightarrow{(e,\mathcal{E})} C'$

$\equiv$ {Def. 2.7}

$\exists L \neq \emptyset.\ H \uparrow \emptyset :: (C, \{e\}) \xrightarrow{L} (C', \mathcal{E})$

$\equiv$ {Rule 1 of Def. 2.8}

$C \xrightarrow{(e,\mathcal{E})}_{\Sigma} C'$

Part ii -

$\nexists C' \in \mathrm{Conf}_H, \mathcal{E} \in \Theta_E.\ C \xrightarrow{(e,\mathcal{E})} C'$

$\equiv$ {Def. 2.7; Logics}

$\nexists L \subseteq \mathcal{T}\ H, C' \in \mathrm{Conf}_H, \mathcal{E} \in \Theta_E.\ H \uparrow \emptyset :: (C, \{e\}) \xrightarrow{L} (C', \mathcal{E})$

$\vee$

$\exists C' \in \mathrm{Conf}_H, \mathcal{E} \in \Theta_E.\ H \uparrow \emptyset :: (C, \{e\}) \xrightarrow{\emptyset} (C', \mathcal{E})$

$\equiv$ {Lemma A.1 (i) and (ii)}

$H \uparrow \emptyset :: (C, \{e\}) \xrightarrow{\emptyset} (C, nil)$

$\Rightarrow$ {Rule 2 of Def. 2.8}

$\Leftarrow$ {Lemma A.1 (ii)}

$C \xrightarrow{(e,\Sigma)}_{\Sigma} C$ □

D.3 Proof of Lemma 2.2

Part i -

Follows directly from Lemma A.1 (i)

Part ii -

$C \xrightarrow{(e,\Sigma)}_{\Sigma} C'$

$\Rightarrow$ {Rule 2 of Def. 2.8}

$H \uparrow \emptyset :: (C, \{e\}) \xrightarrow{\emptyset} (C', \mathcal{E})$

$\Rightarrow$ {Lemma A.1 (ii)}

$C = C'$

Part iii -

By contradiction. Suppose there exist $C' \in \mathrm{Conf}_H$ and $\mathcal{E} \in \Theta_E$ such that $C \xrightarrow{(e,\mathcal{E})}_{\Sigma} C'$ while also $C \xrightarrow{(e,\Sigma)}_{\Sigma} C$. By rule 1 of Def. 2.8, $C \xrightarrow{(e,\mathcal{E})}_{\Sigma} C'$ would imply $H \uparrow \emptyset :: (C, \{e\}) \xrightarrow{L} (C', \mathcal{E})$ for some $L \neq \emptyset$. But from $C \xrightarrow{(e,\Sigma)}_{\Sigma} C$, by rule 2 of Def. 2.8, we would also have $H \uparrow \emptyset :: (C, \{e\}) \xrightarrow{\emptyset} (C'', \mathcal{E}')$, which, using Lemma A.1 (ii) and (iii), would lead to $\nexists L \neq \emptyset.\ H \uparrow \emptyset :: (C, \{e\}) \xrightarrow{L}$, which is a contradiction. □

D.4 Proof of Lemma 2.3

Part i -
Suppose ${}^{\Sigma}\mathcal{C} \xrightarrow{\gamma}_{\Sigma} {}^{\Sigma}\mathcal{C}'$. Without loss of generality assume $\gamma = \alpha_0\beta_0\alpha_1\beta_1\ldots\alpha_n\beta_n$ with $\alpha_i \in \mathrm{IOP}_E^*, \beta_i \in (E\times\{\Sigma\})^*$ for $i = 0,\ldots,n$. In particular, this means that there exist ${}^{\Sigma}\mathcal{C}_0, {}^{\Sigma}\mathcal{C}_1, \ldots, {}^{\Sigma}\mathcal{C}_{n+1}$ such that

$$ {}^{\Sigma}\mathcal{C}_0 = {}^{\Sigma}\mathcal{C},\ {}^{\Sigma}\mathcal{C}_{n+1} = {}^{\Sigma}\mathcal{C}'$$

$$ {}^{\Sigma}\mathcal{C}_0 \xrightarrow{\alpha_0}_{\Sigma} {}^{\Sigma}\mathcal{C}_1 \xrightarrow{\beta_0}_{\Sigma} {}^{\Sigma}\mathcal{C}_1 \xrightarrow{\alpha_1}_{\Sigma} \ldots \xrightarrow{\beta_{n-1}}_{\Sigma} {}^{\Sigma}\mathcal{C}_n \xrightarrow{\alpha_n}_{\Sigma} {}^{\Sigma}\mathcal{C}_{n+1} \xrightarrow{\beta_n}_{\Sigma} {}^{\Sigma}\mathcal{C}_{n+1}$$

Notice that ${}^{\Sigma}\mathcal{C}_i \xrightarrow{\beta_{i-1}}_{\Sigma} {}^{\Sigma}\mathcal{C}_i$ i.e., the configuration does not change by performing β_{i-1}, due to Lemma 2.2(ii). Thus, from the above sequence of transitions we easily get the following one:

$$ {}^{\Sigma}\mathcal{C}_0 \xrightarrow{\alpha_0}_{\Sigma} {}^{\Sigma}\mathcal{C}_1 \xrightarrow{\alpha_1}_{\Sigma} \ldots \xrightarrow{\alpha_{n-1}}_{\Sigma} {}^{\Sigma}\mathcal{C}_n \xrightarrow{\alpha_n}_{\Sigma} {}^{\Sigma}\mathcal{C}_{n+1}$$

But then, noting that $\gamma\setminus\Sigma = \alpha_0\alpha_1\ldots\alpha_n \in \mathrm{IOP}_E^*$ and using Corollary 2.1, we get easily $\mathcal{C} \xrightarrow{\gamma\setminus\Sigma} \mathcal{C}'$
Part ii -

$\gamma \in (L\ {}^{\Sigma}\mathcal{C})$

$\Rightarrow$ {Def. of L}

$\exists \mathcal{C}'.\ {}^{\Sigma}\mathcal{C} \xrightarrow{\gamma}_{\Sigma} {}^{\Sigma}\mathcal{C}'$

$\Rightarrow$ {Part (i) of this lemma}

$\exists \mathcal{C}'.\ (\mathcal{C} \xrightarrow{\gamma\setminus\Sigma} \mathcal{C}')$

$\Rightarrow$ {Corollary 2.1; $\gamma\setminus\Sigma \in \mathrm{IOP}_E^*$}

$\exists \mathcal{C}'.\ (\mathcal{C} \xrightarrow{\gamma\setminus\Sigma} \mathcal{C}') \wedge ({}^{\Sigma}\mathcal{C} \xrightarrow{\gamma\setminus\Sigma}_{\Sigma} {}^{\Sigma}\mathcal{C}')$

$\Rightarrow$ {Def. of L; Logics}

$\gamma\setminus\Sigma \in (L\ \mathcal{C}) \cap (L\ {}^{\Sigma}\mathcal{C})$

Part iii -
Directly follows from Corollary 2.1. □

D.5 Proof of Lemma 4.1

$\gamma \in (L\ s)\ \wedge\ i \in \mathcal{I}\ \wedge\ u \in \mathtt{OUT}\ s'\ \gamma\ i$

$\Rightarrow$ {Lemma 2.1}

$\gamma(i,u) \in (L\ s')$

$\Rightarrow$ {$(L\ s') \subseteq (L\ s)$ by hypothesis}

$\gamma(i,u) \in (L\ s)$

$\Rightarrow$ {Lemma 2.1}

$u \in \mathtt{OUT}\ s\ \gamma\ i$ □

D.6 Proof of Lemma 4.2

By contradiction; suppose that there exists $\gamma \in (L\ \ s') \setminus (L\ \ s)$. Let $\bar{\gamma}$ the longest prefix of γ such that $\bar{\gamma} \in (L\ \ s)$; such a prefix exists since at least $\epsilon \in (L\ \ s)$ by definition. Let us assume $\gamma = \bar{\gamma}(i,u)\gamma'$ for some $i \in \mathcal{I}', u \in \mathcal{U}', \gamma' \in (\mathcal{I}'\times\mathcal{U}')^*$. We can now derive the following:

$\bar{\gamma}(i,u) \in (L\ s')$

$\Rightarrow$ {Lemma 2.1}

$u \in \mathtt{OUT}\ s'\ \bar{\gamma}\ i$

$\Rightarrow$ {$\bar{\gamma} \in (L\ s)$ by assumption; $i \in \mathcal{I}'$ by assumption; $\mathcal{I}' \subseteq \mathcal{I}$ and $s' \sqsubseteq_{\mathtt{co}} s$ by hypothesis}

$u \in \mathtt{OUT}\ s\ \bar{\gamma}\ i$

$\Rightarrow$ {Lemma 2.1}

$\bar{\gamma}(i,u) \in (L\ s)$

which is a contradiction since $\bar{\gamma}(i,u)$ is prefix of γ. □

D.7 Proof of Lemma 5.1

We prove L (**LTS** H') $\subseteq L$ (**LTS** H), that is (**LTS** H') $<<_{MAY}$ (**LTS** H), which, due to Theorem 3.1 is equivalent to (**LTS** H') $\sqsubseteq_{\sim_{MAY}}$ (**LTS** H).

$\gamma \in L$ (**LTS** H')

$\Rightarrow$ {Lemma 2.3 (iii)}

$\gamma \in L$ ($^{\Sigma}$**LTS** H')

$\Rightarrow$ {Lemma 4.2; ($^{\Sigma}$**LTS** H') $\sqsubseteq_{\mathtt{co}}$ ($^{\Sigma}$**LTS** H) and $E' \subseteq E$ by hypothesis}

$\gamma \in L$ ($^{\Sigma}$**LTS** H)

$\Rightarrow$ {Lemma 2.3 (ii); $\gamma \in L$ (**LTS** H') implies $\gamma \setminus \Sigma = \gamma$}

$\gamma \in L$ (**LTS** H) □

D.8 Proof of Lemma 5.2

We let $^{\Sigma}\mathcal{C}' = (^{\Sigma}\mathbf{LTS}\ H'), ^{\Sigma}\mathcal{C} = (^{\Sigma}\mathbf{LTS}\ H), \mathcal{C}' = (\mathbf{LTS}\ H')$ and $\mathcal{C} = (\mathbf{LTS}\ H)$. We proceed by contradiction. Suppose $^{\Sigma}\mathcal{C}' \sqsubseteq_{\mathtt{co}} {}^{\Sigma}\mathcal{C}$ and $\mathcal{C} \not\sqsubseteq_{MUST} \mathcal{C}'$, that is there is an experimenter $\mathcal{O}$ such that $\perp \notin \mathrm{Result}(\mathcal{O},\mathcal{C})$ and $\perp \in \mathrm{Result}(\mathcal{O},\mathcal{C}')$. Let $\mathbf{C} = \mathcal{O} \parallel \mathcal{C}' \rightsquigarrow \ldots$ be an unsuccessful computation in $\mathrm{Comp}(\mathcal{O},\mathcal{C}')$. We distinguish two cases according to $\mathbf{C}$.

Case 1: $\mathbf{C}$ is finite
W.l.g. let us assume $\mathbf{C} = \mathcal{O}_0 \parallel \mathcal{C}'_0 \rightsquigarrow \ldots \rightsquigarrow \mathcal{O}_n \parallel \mathcal{C}'_n$ with $\mathcal{O}_0 = \mathcal{O}$, $\mathcal{C}'_0 = \mathcal{C}'$ and there exist no $\mathcal{O}_{n+1} \parallel \mathcal{C}'_{n+1}$ such that $\mathcal{O}_n \parallel \mathcal{C}'_n \rightsquigarrow \mathcal{O}_{n+1} \parallel \mathcal{C}'_{n+1}$. In this case we have a derivation $\mathcal{C}'_0 \xrightarrow{\gamma} \mathcal{C}'_n$ on the side of $\mathcal{C}'$, with $\gamma = (e_1,\mathcal{E}_1)\ldots(e_k,\mathcal{E}_k) \in \mathrm{IOP}^*_E$, and a sequence of transitions $\mathcal{O}_j \xrightarrow{\mu_j} X_j$, for $j = 1 \ldots n-1$, such that either $\mu_j = e_i$ and $\mathcal{O}_{j+1} = (X_j\ \mathcal{E}_i)$, for some i with $1 \leq i \leq k$, or $\mu_j = \tau$ and $\mathcal{O}_{j+1} = X_j$. Notice that the derivation on the side of the experimenter involves a sequence γ' which is equal to γ up to τ moves.

First of all, notice that it cannot be $\mathcal{O}_n \xrightarrow{\tau}$ since otherwise $\mathbf{C}$ could not be a computation, not being maximal. There are two other possibilities left[4] :

Case 1.1: $\forall e \in E.\ \mathcal{O}_n \xrightarrow{e}\!\!\Rightarrow \not\exists \mathcal{E} \in \Theta_E.\ \mathcal{C}'_n \xrightarrow{(e,\mathcal{E})}$

[4]Notice that $E \subseteq E'$ is necessary otherwise considering only cases 1.1 and 1.2. would not be enough. In particular, it could be the case that $\mathcal{O}_n \xrightarrow{e'}$ for some $e' \in E \setminus E'$ (notice that in this case $\mathcal{O}$ should be over $E \cup E'$) and $\mathcal{O}_n \not\xrightarrow{e}$ for all $e \in E'$. This would mean that $\mathbf{C}$ would be maximal but we could not infer $^{\Sigma}\mathcal{C} \xrightarrow{(e',\Sigma)}_{\Sigma}$ and in fact it could very well be $^{\Sigma}\mathcal{C} \xrightarrow{(e',\mathcal{E}')}_{\Sigma}$ for some $\mathcal{E}' \in \Theta_E$ and this extra step could bring to success so we would not reach contradiction.

By Lemma 5.1[5] we know that $\gamma \in L\ \mathcal{C}$ since $\gamma \in L\ \mathcal{C}'$, by definition of $L\ \mathcal{C}'$ and ${}^{\Sigma}\mathcal{C}' \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathcal{C}$ by hypothesis. Moreover, by Lemma 2.3 (iii), we also know that $\gamma \in L\ {}^{\Sigma}\mathcal{C}$. Thus, again by $\mathcal{C}' \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathcal{C}$ we get

$$\mathtt{OUT}\ {}^{\Sigma}\mathcal{C}'\ \gamma\ e \subseteq \mathtt{OUT}\ {}^{\Sigma}\mathcal{C}\ \gamma\ e \tag{3}$$

since $\gamma \in L\ {}^{\Sigma}\mathcal{C}$ and $e \in E$[6].

Moreover, by hypothesis we know that there is no $\mathcal{E}$ such that $\mathcal{C}'_n \xrightarrow{(e,\mathcal{E})}$, so, by Theorem 2.3, we also know that ${}^{\Sigma}\mathcal{C}'_n \xrightarrow{(e,\Sigma)}_{\Sigma}$. Moreover, ${}^{\Sigma}\mathcal{C}' \xrightarrow{\gamma}_{\Sigma} {}^{\Sigma}\mathcal{C}'_n$, by Corollary 2.1, since $\mathcal{C}' \xrightarrow{\gamma} \mathcal{C}'_n$ and $\gamma \in \mathrm{IOP}^*_E$. By definition of OUT, we get $\Sigma \in \mathtt{OUT}\ {}^{\Sigma}\mathcal{C}'\ \gamma\ e$ so, using relation (3) above we can conclude $\Sigma \in \mathtt{OUT}\ {}^{\Sigma}\mathcal{C}\ \gamma\ e$. But then, again by definition of OUT, we derive that there exists ${}^{\Sigma}\mathcal{C}_n$ such that ${}^{\Sigma}\mathcal{C} \xrightarrow{\gamma}_{\Sigma} {}^{\Sigma}\mathcal{C}_n$ and ${}^{\Sigma}\mathcal{C}_n \xrightarrow{(e,\Sigma)}_{\Sigma}$, and again by Theorem 2.3 and its corollary we get $\mathcal{C}_n \xrightarrow{(e,\mathcal{E})}$ for no $\mathcal{E} \in \Theta_E$ and $\mathcal{C} \xrightarrow{\gamma} \mathcal{C}_n$. This means that we can build the following computation $\mathcal{O}_0 \mid\mid \mathcal{C}_0 \rightsquigarrow \ldots \mathcal{O}_n \mid\mid \mathcal{C}_n$, with $\mathcal{C}_0 = \mathcal{C}$, which is an unsuccessful computation since $\mathbf{C}$ above was so. This contradicts $\perp \notin \mathrm{Result}(\mathcal{O}, \mathcal{C})$

Case 1.2: $\mathcal{O}_n \xrightarrow{e}$ for no $e \in E$
By Lemma 5.1 we know that $\gamma \in L\ \mathcal{C}$ since $\gamma \in L\ \mathcal{C}'$ and ${}^{\Sigma}\mathcal{C}' \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathcal{C}$. This means, by definition of $L\ \mathcal{C}$, that $\mathcal{C} \xrightarrow{\gamma} \mathcal{C}_n$ for some $\mathcal{C}_n$. But then we can build the following computation $\mathcal{O}_0 \mid\mid \mathcal{C}_0 \rightsquigarrow \ldots \mathcal{O}_n \mid\mid \mathcal{C}_n$, with $\mathcal{C}_0 = \mathcal{C}$, which is an unsuccessful computation since $\mathbf{C}$ above was so. This contradicts $\perp \notin \mathrm{Result}(\mathcal{O}, \mathcal{C})$

Case 2: $\mathbf{C}$ is infinite
Also in this case the computation gives raise to one derivation on the side of $\mathcal{C}'$, $\mathcal{C}' \xrightarrow{\gamma}$ and to a sequence of transitions $\mathcal{O}_j \xrightarrow{\mu_j} X_j$ on the side of $\mathcal{O}$, which involve infinite string $\gamma' \in \mathrm{IOP}^{\infty}_E$, which is equal to γ up to τ moves. We distinguish two cases:

Case 2.1: $\forall n \geq 0.\ \exists m \geq n.\ \gamma'_{\downarrow m} \neq \tau$
From the operational semantics rules of experimental systems (Def. 2.10) we get that also γ is infinite. For each finite prefix $\bar{\gamma}$ of γ, by Lemma 5.1 we get $\bar{\gamma} \in L\ \mathcal{C}$ since ${}^{\Sigma}\mathcal{C}' \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathcal{C}$ by hypothesis. By definition of $L\ \mathcal{C}$ this means $\mathcal{C} \xrightarrow{\bar{\gamma}}$. Thus we can build infinite computation $\mathcal{O}_0 \mid\mid \mathcal{C}_0 \rightsquigarrow \ldots$, with $\mathcal{C}_0 = \mathcal{C}$, using, in each step j exactly the same $\mathcal{O}_j$ appearing in $\mathbf{C}$ and the same prefix of γ on which $\mathbf{C}$ is **r.o.** up to step j. Notice also that there can be more than one successive steps using the same prefix $\bar{\gamma}$ due to the fact that $\mathcal{O}_j \xrightarrow{\tau}$ may hold. In conclusion, also in this case we reach a contradiction since the computation we can build involves the same $\mathcal{O}_j$, for $j \geq 0$, occurring in $\mathbf{C}$, which is unsuccessful.

Case 2.2: $\exists n \geq 0.\ \forall m \geq n.\ \gamma'_{\downarrow m} = \tau$
In this case γ is finite and by Lemma 5.1 we get $\mathcal{C} \xrightarrow{\gamma}$. Therefore we can build an unsuccessful computation $\mathcal{O}_0 \mid\mid \mathcal{C}_0 \rightsquigarrow \ldots$ as in Case 2.1 reaching a contradiction. □

D.9 Proof of Lemma 5.3

$({}^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\sim_{MAY}} ({}^{\Sigma}\mathbf{LTS}\ H')$

$\equiv$ {Theorem 3.1}

$({}^{\Sigma}\mathbf{LTS}\ H)\ <<_{MAY}\ ({}^{\Sigma}\mathbf{LTS}\ H')$

$\equiv$ {Def. 3.7}

$L\ ({}^{\Sigma}\mathbf{LTS}\ H) \subseteq L\ ({}^{\Sigma}\mathbf{LTS}\ H')$

$\Rightarrow$ {Lemma 4.1}

[5] Here we need $E' \subseteq E$.

[6] Notice that we can say $e \in E$ because $E = E'$. Otherwise the hypothesis would be $e \in E'$ and here we would need again $E' \subseteq E$.

$\Leftarrow$ {Lemma 4.2; $E \subseteq E'$}

$(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\mathbf{co}} (\mathbf{LTS}\ H')$ □

D.10 Proof of Lemma 5.4

$(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\sim_{MUST}} (^{\Sigma}\mathbf{LTS}\ H')$

$\Rightarrow$ {Lemma 3.2}

$L\ (^{\Sigma}\mathbf{LTS}\ H') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Lemma 4.1}

$(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathbf{co}} (^{\Sigma}\mathbf{LTS}\ H)$ □

D.11 Proof of Theorem 5.1

Part i -

$(^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\sim_{MAY}} (^{\Sigma}\mathbf{LTS}\ H') \wedge (^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathbf{co}} (^{\Sigma}\mathbf{LTS}\ H)$

$\equiv$ {Theorem 3.1}

$(^{\Sigma}\mathbf{LTS}\ H'') <<_{MAY} (^{\Sigma}\mathbf{LTS}\ H') \wedge (^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathbf{co}} (^{\Sigma}\mathbf{LTS}\ H)$

$\equiv$ {Def. 3.7}

$L\ (^{\Sigma}\mathbf{LTS}\ H'') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H') \wedge (^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathbf{co}} (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Lemma 4.2}

$L\ (^{\Sigma}\mathbf{LTS}\ H'') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H') \wedge L\ (^{\Sigma}\mathbf{LTS}\ H') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Set Theory}

$L\ (^{\Sigma}\mathbf{LTS}\ H'') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Lemma 4.1}

$(^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\mathbf{co}} (^{\Sigma}\mathbf{LTS}\ H)$

Part ii -

$(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\sim_{MUST}} (^{\Sigma}\mathbf{LTS}\ H'') \wedge (^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathbf{co}} (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Lemma 3.2}

$L\ (^{\Sigma}\mathbf{LTS}\ H'') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H') \wedge (^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathbf{co}} (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Lemma 4.2}

$L\ (^{\Sigma}\mathbf{LTS}\ H'') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H') \wedge L\ (^{\Sigma}\mathbf{LTS}\ H') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Set Theory}

$L\ (^{\Sigma}\mathbf{LTS}\ H'') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Lemma 4.1}

$(^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\mathbf{co}} (^{\Sigma}\mathbf{LTS}\ H)$

Part iii -

Directly follows from Part (ii) and the fact that $\sqsubseteq_{\sim}$ is stronger than $\sqsubseteq_{\sim_{MUST}}$ by Def. 3.6.

Part iv -

$(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{co} (^{\Sigma}\mathbf{LTS}\ H) \wedge (^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\simeq_{MAY}} (^{\Sigma}\mathbf{LTS}\ H'')$

$\Rightarrow$ {Lemma 4.2, $E' \subseteq E$}

$L\ (^{\Sigma}\mathbf{LTS}\ H') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H) \wedge (^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\simeq_{MAY}} (^{\Sigma}\mathbf{LTS}\ H'')$

$\Rightarrow$ {Theorem 3.1;Def. 3.7}

$L\ (^{\Sigma}\mathbf{LTS}\ H') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H) \wedge L\ (^{\Sigma}\mathbf{LTS}\ H) \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H'')$

$\Rightarrow$ {Set Theory}

$L\ (^{\Sigma}\mathbf{LTS}\ H') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H'')$

$\Rightarrow$ {Lemma 4.1}

$(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{co} (^{\Sigma}\mathbf{LTS}\ H'')$

Part v -

$(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{co} (^{\Sigma}\mathbf{LTS}\ H) \wedge (^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\simeq_{MUST}} (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Lemma 4.2, $E' \subseteq E$}

$L\ (^{\Sigma}\mathbf{LTS}\ H') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H) \wedge (^{\Sigma}\mathbf{LTS}\ H'') \sqsubseteq_{\simeq_{MUST}} (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Lemma 3.2}

$L\ (^{\Sigma}\mathbf{LTS}\ H') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H) \wedge L\ (^{\Sigma}\mathbf{LTS}\ H) \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H'')$

$\Rightarrow$ {Set Theory}

$L\ (^{\Sigma}\mathbf{LTS}\ H') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H'')$

$\Rightarrow$ {Lemma 4.1}

$(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{co} (^{\Sigma}\mathbf{LTS}\ H'')$

Part vi -

Directly follows from Part (v) and the fact that $\sqsubseteq_{\simeq}$ is stronger than $\sqsubseteq_{\simeq_{MUST}}$ by Def. 3.6. □

D.12 Proof of Lemma 5.5

$(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\simeq_{MAY}} (^{\Sigma}\mathbf{LTS}\ H')$

$\equiv$ {Theorem 3.1}

$(^{\Sigma}\mathbf{LTS}\ H)\ <<_{MAY}\ (^{\Sigma}\mathbf{LTS}\ H')$

$\equiv$ {Def. 3.7}

$L\ (^{\Sigma}\mathbf{LTS}\ H) \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H')$

$\Rightarrow$ {$L\ (\mathbf{LTS}\ H) \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H)$ by Lemma 2.3 (iii)}

$L\ (\mathbf{LTS}\ H) \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H')$

$\Rightarrow$ {$(L\ (\mathbf{LTS}\ H)) \setminus \Sigma = L\ (\mathbf{LTS}\ H)$; Lemma 2.3 (ii)}

$L\ (\mathbf{LTS}\ H) \subseteq L\ (\mathbf{LTS}\ H')$

$\equiv$ {Def. 3.7}

$(\mathbf{LTS}\ H)\ <<_{MAY}\ (\mathbf{LTS}\ H')$

$\equiv$ {Theorem 3.1}

$(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\sim MAY} (^{\Sigma}\mathbf{LTS}\ H')$ □

D.13 Proof of Lemma 5.6

$(^{\Sigma}\mathbf{LTS}\ H) \sqsubseteq_{\sim MUST} (^{\Sigma}\mathbf{LTS}\ H')$

$\Rightarrow$ {Lemma 3.2}

$L\ (^{\Sigma}\mathbf{LTS}\ H') \subseteq L\ (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Lemma 4.1}

$(^{\Sigma}\mathbf{LTS}\ H') \sqsubseteq_{\mathsf{co}} (^{\Sigma}\mathbf{LTS}\ H)$

$\Rightarrow$ {Lemma 5.2[7]}

$(\mathbf{LTS}\ H) \sqsubseteq_{\sim MUST} (\mathbf{LTS}\ H')$ □

[7] The use of Lemma 5.2 requires $E = E'$