Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Logic and
Algebraic Programming 69 (2006) 1–74

THE JOURNAL OF
LOGIC AND
ALGEBRAIC
PROGRAMMING

www.elsevier.com/locate/jlap

On testing UML statecharts ☆

Mieke Massink, Diego Latella *, Stefania Gnesi

CNR/Istituto di Scienza e Tecnologie dell'Informazione "A Faedo", via Moruzzi 1, I56124 Pisa, Italy

Received 15 April 2005; received in revised form 8 March 2006; accepted 8 March 2006

Abstract

We present a formal framework for notions related to testing and model based test generation for a behavioural subset of UML Statecharts (UMLSCs). This framework builds, on one hand, upon formal testing and conformance theory that has originally been developed in the context of process algebras and Labeled Transition Systems (LTSs), and, on the other hand, upon our previous work on formal semantics for UMLSCs. The paper covers the development of proper extensional testing preorders and equivalence for UMLSCs. We present an algorithm for testing equivalence verification which is based on an intensional characterization of the testing relations. Testing equivalence verification is reduced to bisimulation equivalence verification. We also address the issue of conformance testing and present a formal conformance relation together with a test case generation algorithm which is proved sound and exhaustive w.r.t. the conformance relation. We show results on the formal relationship of the testing relations with the conformance one. The comprehensive and uniform approach presented in this paper sets the theoretical basis for UMLSCs testing frameworks and makes them available for practitioners in industry where the UML has become a de facto standard, in particular there where it is used for the development of complex concurrent systems.

Keywords: UML statecharts; Formal testing theories; Testing equivalence; Mechanical verification; Formal conformace testing; Test case mechanical generation; Formal semantics

1. Introduction

Modern societies strongly depend, for their functioning as well as for the protection of their citizens, on systems of highly interconnected and interdependent infrastructures,

☆ The work presented in the present paper has been partially funded by project EU-IST-3-016004-IP-09 (SENSORIA).

* Corresponding author. Tel.: +39 0503152982.

E-mail addresses: mieke.massink@isti.cnr.it (M. Massink), diego.latella@isti.cnr.it (D. Latella), stefania.gnesi@isti.cnr.it (S. Gnesi).

1567-8326/$ - see front matter
doi:10.1016/j.jlap.2006.03.001

which are increasingly based on computer systems and software. The complexity of such systems, and those of the near future, will be higher than that of any artifact which has been built so far. In recent years, the Unified Modeling Language (UML) [34] has been introduced as a notation for modeling and reasoning about large and complex systems, and their design, across a wide range of application domains. Moreover system modeling and analysis techniques, especially those based on formal methods, are more and more used for enhancing traditional System Engineering techniques for improving system quality. In particular this holds for testing and model-based formal test case derivation using formal conformance testing.

Testing and conformance relations in the context of labeled transition systems (LTSs) have been thoroughly investigated in the literature. Broadly speaking, 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) at hand. An account of the major results in the area of testing and conformance relations can be found in [9,17,40–42]. The theory has been developed mainly in the context of process algebras and input/output transition systems.

In this paper we present a uniform, formal, approach to a *testing theory* and *equivalence* as well as *conformance testing* and *test case generation* for UML Statecharts[1] (UMLSCs, for short), based on our previous work presented in [25,26,16].

The UML consists of a number of diagrammatic specification languages, among which UMLSCs, that are intended for the specification of behavioral aspects of software systems. This diagrammatic notation differs considerably from process algebraic notations. In UMLSCs, transitions are labeled by input/output-*pairs* (i/o-pairs), where the relation between input and output is maintained at the level of the *single* transitions. This is neither the case in traditional testing theories, like [17], where no distinction is made between input and output, nor for the input/output transition systems used in standard conformance testing theory [42]. In our approach we use transition systems labeled over i/o-pairs where a generic transition models a *step* of the associated statechart (*step*-transition), thus preserving the atomicity of input acquisition and output generation in a single step. The advantages of such a semantic model choice are discussed in [16], where the interested reader is referred to.

In [25] a general testing theory for UMLSCs has been defined using a framework similar to that proposed in [30], which was in turn inspired by the work of Hennessy for traditional LTSs [17]. The general approach of the above-mentioned theories is based on the well known notions of *MAY* and *MUST* preorders and related equivalences. Intuitively, for systems A and B, $A \sqsubseteq_{MAY} B$ means that if a generic experimenter (i.e. test case) E has *a* successful test run while testing A, then E has also *a* successful test run when testing B. On the other hand, $A \sqsubseteq_{MUST} B$ means that if *all* test runs of a generic experimenter E are successful when testing A, then it must be the case that *all* test runs of E are successful when testing B. It can be shown that $\sqsubseteq_{MAY}$ coincides with trace inclusion and that $A \sqsubseteq_{MUST} B$ implies $B \sqsubseteq_{MAY} A$. Thus, the testing preorders focus essentially on the observable behavior of systems and are strongly related to their internal non-determinism and deadlock capabilities; intuitively, if both $A \sqsubseteq_{MAY} B$ and $A \sqsubseteq_{MUST} B$ hold, then A is "more non-deterministic" than B and can generate more deadlocks than B can, when tested by an experimenter. Finally, if also the reverse preorders hold, i.e. $B \sqsubseteq_{MAY} A$ and $B \sqsubseteq_{MUST} A$ as well, then A and B are *testing*

[1] Although we refer to UML 1.5, the main features of the notation of interest for our work have not changed in later versions.

equivalent since no experimenter can distinguish them. The main semantic assumptions in [17] are that (i) system interaction is modeled by action-synchronization rather than input/output exchanges, and (ii) absence of reaction from a system to a stimulus presented by an experimenter results in a deadlock affecting both the system and the experimenter. In [30], and later in [25] specifically for UMLSCs, assumption (i) has been replaced by modeling system interaction as input/output exchanges, but assumption (ii) remained unchanged. In particular, in [25], absence of reaction of a given state s on a given input i is represented by the absence of *any* transition with such an input i from s, in a way which is typical of the process-algebraic approach. We refer to the resulting semantic model as the "non-stuttering" one. The testing equivalence verification algorithm has originally been developed on the non-stuttering semantics and exploits an intensional characterization of testing preorders/equivalence. More specifically, Functional Acceptance Automata (FAAs), a compact representation of the LTSs associated to UMLSCs by their semantics, is defined using the intensional characterization of testing relations. It is shown that testing equivalence of UMLSCs coincides with bisimulation equivalence of (a variant of) their associated FAAs. The algorithm translates the LTSs associated to the UMLSCs into FAAs and checks their bisimulation equivalence after proper manipulation of their labels. It is partially based on results proposed in [4].

In [16] we proposed a formal conformance testing relation and a test case generation algorithm for *input enabled* labeled transition systems over i/o-pairs, i.e. LTSs where each state has (at least) one outgoing transition for each element of the input alphabet of the transition system. Intuitively, such transition systems cannot refuse any of the specified input events, in the sense that they cannot deadlock when such events are offered to them by the external environment. Whenever a machine, in a given state, *does not* react on a given input, its modeling LTS has a specific loop-transition from the corresponding state to itself, labeled by that input and a special "stuttering" output-label.

Input enabled LTSs over i/o-pairs have been used as semantic model, which we call the "stuttering semantics", for a behavioral subset of UMLSCs [14], which can be seen as system specifications. Moreover, input enabled LTSs over i/o-pairs are also suitable for modeling implementations of systems specified by such diagrams. Modeling implementations as input enabled LTSs is common practice in the context of formal conformance testing—see e.g. [42]. The conformance relation we defined in [14] is similar to the one of Tretmans [42], with adaptations which take care of our semantic framework for UMLSCs.

The test case generation algorithm we present is both *exhaustive* and *sound* with respect to the conformance relation. Exhaustiveness ensures that if an implementation passes all test cases generated by the algorithm from a given specification, then it conforms to the specification. Conversely, soundness ensures that if an implementation conforms to a specification, then it passes all test cases generated by the algorithm from such a specification. The testing equivalence verification algorithm naturally extends to the case of stuttering semantics.

The two different ways of dealing with absence of reaction, and in particular, the ability for experimenters to explicitly *detect* absence of reaction turns out to be of major importance for determining the relative expressive power of the various semantics. More specifically, we define *MAY* and *MUST* preorders also for the stuttering semantics and we provide a formal comparison between the Hennessy-like, non-stuttering semantics [25,17], and the stuttering semantics w.r.t. testing and conformance ordering relations; we show that if two UMLSCs, say A and B, are in conformance relation (i.e. A conforms to B) in the stuttering semantics, then they are also in *MAY* and in the reverse-*MUST* relations (i.e. $A \sqsubseteq_{MAY} B$

and $B \sqsubseteq_{MUST} A$) in the non-stuttering semantics, *but not vice-versa*. This shows that the Hennessy-like, non-stuttering, semantics [25,17] is not adequate for reasoning about issues of conformance, since the detection of absence of reaction, explicitly modeled only in the stuttering semantics, plays a major role when dealing with conformance. Accordingly, the following results are proved: *in the stuttering semantics*, the conformance relation essentially coincides with the *MAY* preorder, and is strictly weaker than the testing preorder. Moreover, in the stuttering semantics, nice substitutivity properties hold; for instance, testing equivalent implementations conform to the same specifications and implementations conform to testing equivalent specifications. The above results have been originally presented in [26] and in the present paper we include all related proofs.

As an additional result of our work, we also defined a specific test case language which we use uniformly in the present paper both for what concerns the testing preorders/equivalence and for automatic test case generation as well as what concerns conformance.

The present paper is organized as follows. Section 2 discusses the relationship of the work presented in the present paper with the literature. In Section 3 the major background notions, necessary for the development of the testing and conformance theories are recalled. Section 4 addresses the testing preorders and the equivalence verification algorithm. Section 5 addresses conformance testing and the test case generation algorithm. Section 6 studies the relationships between the two views at the semantics of UMLSCs presented and used in Sections 4 and 5—namely the "non-stuttering" and the "stuttering" semantics—and compares the testing and the conformance relations. Technical details and results on the dynamic semantics of UMLSCs on which our work is based, and in particular their "core semantics" are given in Appendix A. Appendix B contains the detailed formal proofs of all results presented in this paper.

2. Related work

As briefly mentioned in Section 1, the basic work on formal theories for testing, mainly in the context of Process Algebra and LTSs, has been proposed by De Nicola and Hennessy (see e.g. [9,17]). Tretmans addressed more the issues related to conformance testing in a formal, LTS-based, framework [40–42].

The results addressed in the present paper have been originally proposed in [25,26,16], although in isolation, while in the present paper they are dealt with in a uniform framework and notation. Moreover all proofs, some of which were omitted in the above-mentioned papers, are provided in the present paper.

Our LTSs labeled over i/o-pairs are very similar to Finite State Machines (FSMs), in particular Mealy Machines. A considerable number of studies in the field of testing FSMs are available in the literature. An excellent survey can be found in [28]. Many such proposals deal with test case generation but mainly in the context of *deterministic* machines, as, e.g., in the seminal work of Chow [38], or in [11], where practical applicability of model-based test case generation is addressed. In some proposals, like the one in [5], further restrictions on the machines are introduced, requiring e.g. that they must be strongly connected. Non-determinism in the context of conformance testing FSMs is addressed in [35], where specifications may be non-deterministic, while implementations are required to be deterministic. Specifications and implementations are required to share the same input alphabet. Moreover, only so called *observable* non-deterministic FSMs are considered. Observable non-deterministic FSMs are FSMs which cannot produce the same output on the same input while moving to different next states, i.e. if they move from the current state to different next states they must also produce different outputs. Specification FSMs are not required to be completely specified, i.e. there can be states which have no outgoing transition for some input event (the notion of complete specification is the FSMs analogous of input-enabledness in the context of LTSs). A conformance relation on FSMs, called *reduction*, is given which is very similar to the conformance relation we use in the present paper, and a finite test suites generation algorithm is proposed which is complete in the class of all implementations with a given upper bound on the number of states (completeness in the context of FSMs corresponds to eshaustiveness in the context of LTSs). In [18] the methods are extended in such a way that *adaptive* testing is possible, i.e. information is gathered from the output of an implementation under test that can be used for guiding future testing. Neither [35] nor [18] addresses the issue of linking the testing methodology and algorithms they propose to a general framework where the very notion of testing computing devices, its formalization and the equivalence it induces are addressed, as e.g. in [9,17]. Moreover, we think that the restriction to observable FSMs is a rather strong one, if non-deterministic behaviour is to be addressed. In fact, although for each (completely specified) non-observable FSM there exists an equivalent observable one [18], such an equivalence takes into account only the language defined by the machine and not its deadlock properties, which can be of major interest in specific contexts, possibly different from conformance testing (e.g. non-stuttering semantics of UMLSCs). Furthermore UMLSCs can easily violate the observability constraint. Restricting testing theories to deterministic implementations seems also a rather strong limitation, especially in the context of distributed or concurrent implementations. In such a context, non-determinism arising from concurrency cannot be avoided and, in fact, non-determinism is a key notion in the area of formal approaches to system modeling and verification and it is a central notion in traditional concurrency and testing theories for LTSs [19,33,9,17]. Consequently we use generic LTSs over i/o-pairs without any limitation on the form of non-determinism they may exhibit. The restriction to input-enabled LTSs, when we address conformance, adequately models the UMLSCs stuttering phenomenon. Furthermore, the link we provide to testing equivalence, rather than language equivalence, and in particular its definition in terms of experimenters, in the sense of [9,17], brings in—without renouncing to a solid mathematical framework—a strong intuitive support which is sometimes missing in the above-mentioned works on FSMs testing.

In [21] algorithms for test case generation from UML statecharts are presented which cover both control flow issues and data flow ones. As far as flow control is concerned, statecharts are mapped to (extended) FSMs. The semantic framework on which the presented work is based is a flat one, i.e. the hierarchical structure of UMLSCs is not exploited in the definition of the operational semantics. Moreover, the model presented does not take transition priorities into consideration. The relationship with general testing theories for state/transition structures is not addressed.

Further related work on automatic test generation based on UMLSCs has been developed in the context of the Agedis project [39,8]. In that approach a system model, composed of class, object and statechart diagrams is translated into a model expressed in an intermediate format suitable as input for model checking and test generation tools. It follows a pragmatic, industrial approach with a clear focus on the test selection problem, but with less emphasis on UML formal semantics. In contrast, we follow a 'Semantics-first' approach (also) with respect to conformance testing.

Similarly, in [36] emphasis is put primarily on support tool implementation. The semantics of UMLSCs is defined by means of the tool umlout—which generates LTSs with input and output events, in the style of [42]. In particular, no formal definition of such semantics is given in [36]. We already addressed the issues related to the use of LTSs with separate input and output events as a model for UMLSCs semantics.

Other approaches to automatic test generation include [37] that describes the use of the CASE tool AutoFocus. The authors emphasize the need for a formally defined semantics and use state transition diagrams that resemble a subset of the UML-RT, but it seems there is no formal relation between their diagrams and the subset of the UML-RT. Automated test generation has been developed also for classical Harel statechart diagrams, e.g. [3], which semantically differ considerably from UMLSCs (e.g., a different priority schema as well as a different semantics for the input queues are used).

3. Basic notions

In this section we summarize the definitions concerning LTSs, with particular reference to LTSs over input/output-pairs (i/o-pairs, for short), hierarchical automata, experimenters, experimental systems and their computations, which form the basis for the formal semantics of UMLSCs and related testing and conformance notions as presented in [15,25,26,16].

The definition of a sound "basic" kernel of a notation, to be extended only after its main features have been investigated, has already proved to be a valuable and fruitful methodology and is often standard practice in many fields of concurrency theory, like process-algebra. We refer to e.g. [23] for a deeper discussion on such "basic-notation-first" and "semantics-first" versus "full-notation-first" issue. In line with this approach, in the present paper, we consider a subset of UMLSCs, which includes all the interesting conceptual issues related to concurrency in dynamic behavior—like sequentialization, non-determinism and parallelism—as well as UMLSCs specific issues—like state refinement, transition priorities, interlevel/join/fork transitions. We do *not* consider history, action and activity states; we restrict events to signals without parameters (actually we do not interpret events at all); time and change events, object creation and destruction events, and deferred events are not considered neither are branch transitions; also variables and data are not allowed so that actions are required to be just (sequences of) events. We also abstract from entry and exit actions of states. We refer to [27] for object-based extensions of our basic model which include, among others, object management, e.g. object creation/destruction.

In Section 3.1 basic notions related to Labeled Transition Systems are briefly recalled; Hierarchical Automata are shortly described in Section 3.2 while Section 3.3 recalls the major notions related to testing theories.

3.1. Labeled transition systems

The notion of labeled transition system (LTS) is central in the present paper:

Definition 1 (*LTS*). A labeled transition system (LTS) $\mathcal{M}$ is a tuple $(S, s_{\text{in}}, L, \to)$ where S is the *set of states* with $s_{\text{in}} \in S$ being the *initial* state, L is the *set of (transition) labels* and $\to \ \subseteq S \times L \times S$ is the *transition relation* of the LTS.

For $(s, l, s') \in \rightarrow$ we write $s \xrightarrow{l} s'$. The notation $s \xrightarrow{l}$ will be a shorthand for $\exists s'.\ s \xrightarrow{l} s'$. Some standard definitions are given below.[2]

Definition 2 (*Auxiliary definitions for LTSs*). For LTS $\mathcal{M} = (S, s_{\text{in}}, L, \rightarrow)$, $s, s', s'' \in S$, $l \in L, \gamma \in L^*, \omega \in L^\infty$

- The transition relation $\xrightarrow{\gamma}$ over finite sequences is defined in the obvious way, i.e. it is the smallest relation over $S \times L^* \times S$ such that: (a) $s \xrightarrow{\epsilon} s$ and (b) if $s \xrightarrow{\gamma} s'$ and $s' \xrightarrow{l} s''$, then $s \xrightarrow{\gamma l} s''$; in a similar way as for single labels we let $s \xrightarrow{\gamma}$ stand for $\exists s'.\ s \xrightarrow{\gamma} s'$;
- By $s \xrightarrow{\omega}$ we mean that there exists an infinite sequence $s_0 s_1 s_2 \ldots$ of states in S, with $s = s_0$, $\omega = l_0 l_1 \ldots$, such that for all $n \geq 0$ we have $s_n \xrightarrow{l_n} s_{n+1}$;
- The *language* of $\mathcal{M}$ is the set $\mathsf{lan}\ \mathcal{M} =_{df} \{\gamma \in L^* \mid s_{\text{in}} \xrightarrow{\gamma}\}$ of all finite *traces* of $\mathcal{M}$;
- The labels of $\mathcal{M}$ *after* γ is the set $\mathsf{S}\ \mathcal{M}\ \gamma =_{df} \{l \in L \mid s_{\text{in}} \xrightarrow{\gamma l}\}$;
- The *acceptance sets* of $\mathcal{M}$ *after* γ is the set $\mathsf{AS}\ \mathcal{M}\ \gamma =_{df} \{\mathsf{S}\ s\ \epsilon \mid s_{\text{in}} \xrightarrow{\gamma} s\}$;
- $\mathcal{M}$ is *finite* if sets S and $\rightarrow$ are finite;
- $\mathcal{M}$ is *deterministic* if for all $s, s', s'' \in S$ and $l \in L$, whenever $s \xrightarrow{l} s'$ and $s \xrightarrow{l} s''$ we have $s' = s''$. Notice that if $\mathcal{M}$ is deterministic, each $\gamma \in (\mathsf{lan}\ \mathcal{M})$ *uniquely* identifies a state $s \in S$.

In the definition of acceptance sets we have treated state $s \in S$ of LTS $\mathcal{M}$ as a LTS in turn. The set S_s of states of such LTS contains all and only those elements of S which are reachable from s via $\longrightarrow$ (i.e. $S_s =_{df} \{s' \in S \mid \exists \gamma.\ s \xrightarrow{\gamma} s'\}$), the initial state being s; the transition relation of the LTS is $\longrightarrow \cap (S_s \times S_s)$. We will often treat states of LTSs as LTSs in turn, as above. In the rest of this paper we will use LTSs where the labels in L are i/o-pairs, i.e. $L = L_I \times L_U$, for some input set L_I and output set L_U; we will refer to such LTSs as LTSs *over* $L_I \times L_U$. Finally, notice that, as we will see, for the purposes of the present paper, in particular with reference to the results of Section 4, Section 5, and Section 6, it is sufficient to consider finite LTSs over finite label sets.

3.2. Hierarchical automata

As briefly mentioned in Section 1 we use hierarchical automata (HAs) [32] as an abstract syntax for UMLSCs. HAs for UMLSCs have been introduced in [24,15]. The relevant definitions concerning HAs, like their dynamic semantics, are recalled in Appendix A. In this section we recall, informally, only the main notions which are necessary for the understanding of the paper.

---

[2] In this paper we will freely use a functional programming like notation where 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. By $\exists_1 x.\ \ldots$ we mean $\exists x.\ \ldots$ and such an x is unique. For indexed family of sets X_y, with y in index set Y (resp. y satisfying predicate p), we let $\bigcup_{y \in Y} X_y$ (resp. $\bigcup_{y:(p\ y)} X_y$) denote the union over all the sets of the family and we let $\bigcup_{y \in \emptyset} X_y = \emptyset$ (resp. $\bigcup_{y:\text{FALSE}} X_y = \emptyset$). Moreover, for set X, the set of finite (resp. infinite) sequences over X will be denoted by X^* (resp. X^∞); ϵ denotes the empty sequence; for $x \in X$ we let x denote also the sequence in X^* consisting of the single element x, while for $\gamma, \gamma' \in X^*$ we let the juxtaposition $\gamma\gamma'$ of γ with γ' denote their concatenation. Concatenation is extended to infinite sequences with $\gamma\gamma' = \gamma$ when γ is infinite, and $\gamma\gamma'$ defined in the usual way otherwise.

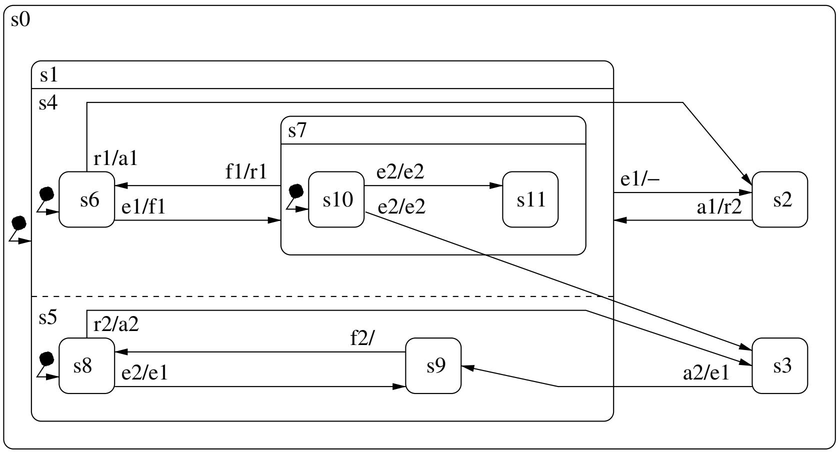


Fig. 1. A sample UMLSC.

Let us consider, as a small example, the UMLSC[3] of Fig. 1 and its corresponding HA in Fig. 2. Roughly speaking, each OR-state of the UMLSC is mapped into a sequential automaton of the HA while basic states and AND-states are mapped into states of the sequential automaton corresponding to the OR-state immediately containing them. Moreover, a refinement function maps each state in the HA corresponding to an AND-state into the set of the sequential automata corresponding to its component OR-states. In the example OR-states $s0$, $s4$, $s5$ and $s7$ are mapped to sequential automata $A0$, $A1$, $A2$ and $A3$, while state $s1$ of $A0$, corresponding to AND-state $s1$ of the UMLSC, is refined into $\{A1, A2\}$. Non-interlevel transitions are represented in the obvious way: for instance transition $t8$ of the HA represents the transition from state $s8$ to state $s9$ of the UMLSC. The labels of transitions are collected in Table 1; for example the *trigger event* of $t8$, namely $EV\ t8$, is $e2$ while its associated *output event*, namely $AC\ t8$ is $e1$. An interlevel transition is represented as a transition t departing from (the HA state corresponding to) its highest source and pointing to (the HA state corresponding to) its highest target. The set of the other sources (resp. targets) are recorded in the *source restriction*—$SR\ t$ (resp. *target determinator*) $TD\ t$, of t. So, for instance, $SR\ t1 = \{s6\}$ means that a necessary condition for $t1$ to be enabled is that the current state configuration contains not only $s1$ (the source of $t1$), but *also* $s6$. Similarly, when firing $t2$ the new state configuration will contain $s6$ and $s8$, besides $s1$. Finally, each transition has a guard $G\ t$, not shown in this example.

Summarizing, basically a HA $H = (F, E, \rho)$ is composed of a finite (non-empty) collection F of sequential automata related by a *refinement function* ρ which imposes on H the hierarchical state nesting-structure of the associated statechart: ρ maps every state s of each automaton in F into a (possibly empty) set of elements of F which refine s. The automata in F are finite, i.e. they have a finite set of states and a finite number of transitions. E is the finite set of events labeling the transitions of the elements of F. For our

[3] In the sequel, we will often omit the names of states in statecharts, when this will not cause confusion. Similarly, we will often refrain from naming the states of automata or LTSs in their graphical representations. Automata (resp. LTSs) states will be drawn as square (resp. circles or ellipses), where initial states will be represented by double square (resp. circles or ellipses). Any information related to a transition, e.g. a label, will usually be placed close to its source.

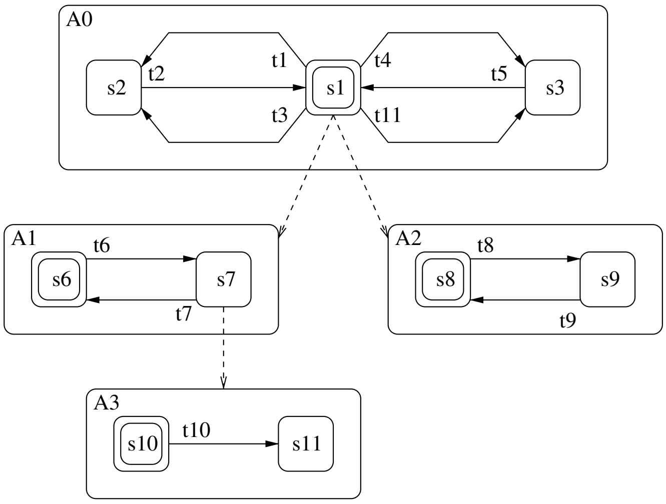


Fig. 2. The HA representing the sample UMLSC of Fig. 1.

Table 1
Transition labels for the HA of Fig. 2

t	$t1$	$t2$	$t3$	$t4$	$t5$	$t6$	$t7$	$t8$	$t9$	$t10$	$t11$
$SR\,t$	$\{s6\}$	$\emptyset$	$\emptyset$	$\{s8\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{s10\}$
$EV\,t$	$r1$	$a1$	$e1$	$r2$	$a2$	$e1$	$f1$	$e2$	$f2$	$e2$	$e2$
$AC\,t$	$a1$	$r2$	ϵ	$a2$	$e1$	$f1$	$r1$	$e1$	ϵ	$e2$	$e2$
$TD\,t$	$\emptyset$	$\{s6, s8\}$	$\emptyset$	$\emptyset$	$\{s6, s9\}$	$\{s10\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

running example, we have $F = \{A0, A1, A2, A3\}$, $E = \{a1, a2, e1, e2, f1, f2, r1, r2\}$, $(\rho\, s1) = \{A1, A2\}$, $(\rho\, s7) = \{A3\}$, and $(\rho\, s) = \emptyset$ otherwise. Inter-level transitions are encoded by means of proper annotations in transition labels. Global states of H, called state *configurations*, are *sets* of states of the automata in F which respect the tree-like structure imposed by ρ. We let Conf_H denote the set of all configurations of H and $\mathcal{C}_{\mathrm{in}}$ its initial configuration, i.e. the configuration composed only of initial states of automata in F.

An issue which deserves to be briefly addressed here is the way in which we deal with the so called *input-queue* of UMLSCs, i.e. their "external environment". In the standard definition of UML statecharts semantics [34], a *scheduler* is in charge of selecting an event from the input-queue of an object, feeding it into the associated state-machine, and letting such a machine produce a step transition. Such a step transition corresponds to the firing of a maximal set of enabled non-conflicting transitions of the statechart associated to the object, provided that certain transition priority constraints are not violated. After such transitions are fired and when the execution of all the actions labeling them is completed, the step itself is completed and the scheduler can choose another event from a queue and start the next cycle. While in classical statecharts the external environment is modeled by a set, in the definition of UML statecharts, the nature of the input-queue of a statechart is not specified; in particular, the management policy of such a queue is not defined. In our overall approach to UMLSCs semantics definition, we choose *not* to fix any particular semantics, such as set, or multi-set or FIFO-queue etc., but to model the input queue in a policy-independent way, freely using a notion of abstract data types. In the following we assume that for set D,

Θ_D denotes the class 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 manipulating such structures. In particular, in the present paper, we let $\mathsf{Add}\ d\ \mathcal{D}$ denote the structure obtained by inserting element d in structure $\mathcal{D}$ and the predicate $(\mathsf{Sel}\ \mathcal{D}\ d\ \mathcal{D}')$ states that $\mathcal{D}'$ is the structure resulting from selecting d from $\mathcal{D}$; of course, the selection policy depends on the choice for the particular semantics. Similarly, $(\mathsf{Join}\ \mathcal{D}\ \mathcal{D}')$ denotes the structure obtained by merging $\mathcal{D}$ with $\mathcal{D}'$. We assume that if $\mathcal{D}$ is the empty structure, denoted by $\langle\rangle$, then $(\mathsf{Sel}\ \mathcal{D}\ d\ \mathcal{D}')$ yields FALSE for all d and $\mathcal{D}'$. We shall often speak of the *input queue*, or simply *queue*, by that meaning a structure in Θ_D, abstracting from the particular choice for the semantics of Θ_D.

The operational semantics of HA H characterizes the relation $(\mathcal{C}, \mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}', \mathcal{E}')$ between *statuses* and transitions fired during a step. A status is a pair $(\mathcal{C}, \mathcal{E})$ where $\mathcal{C}$ is the current configuration and $\mathcal{E}$ is the current input queue.

$(\mathcal{C}, \mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}', \mathcal{E}')$ means that a step-transition of H can be performed in the current status $(\mathcal{C}, \mathcal{E})$ by firing the transitions belonging to set $\mathcal{L}$ and reaching the new status $(\mathcal{C}', \mathcal{E}')$. The new configuration (resp. input-queue) of H after the step will be $\mathcal{C}'$ (resp. $\mathcal{E}'$). The definition of the step-transition relation is given below:

Definition 3 (*Transition rule*).

$$\frac{\begin{array}{l}(\mathsf{Sel}\ \mathcal{E}\ e\ \mathcal{E}'') \\ H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{\mathcal{L}} (\mathcal{C}', \mathcal{E}')\end{array}}{(\mathcal{C}, \mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}', (\mathsf{Join}\ \mathcal{E}''\mathcal{E}'))}$$

The above definition makes use of the so called *Core Semantics*, i.e. the relation

$$A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}', \mathcal{E}').$$

The role of the Core Semantics is the characterization of the set $\mathcal{L}$ of transitions to be fired, their related output-events, $\mathcal{E}'$ and the resulting configuration $\mathcal{C}'$, when HA A is in status $(\mathcal{C}, \mathcal{E})$, under specific constraints P related to transition priorities. All issues of (event) ordering, concurrency, and non-determinism within single statecharts are dealt with by the Core Semantics. Although essential for the definition of the formal semantics, all the above issues are concerned with an intensional view of the statechart behavior, thus they are technically quite orthogonal to the testing and conformance issues which we address in the present paper and which are intrinsically extensional, and, therefore, the details of the Core Semantics are given in Appendix A.

3.3. Definitions related to testing

In order to model how test cases are performed over systems represented by LTSs over $L_I \times L_U$ we first of all need to formalize the notion of experimenters. An *experimenter* is similar to a transition system where the set of states is partitioned into *output* states and *input* states. Output states may challenge the system which the experimenter is experimenting with—by sending it values in L_I for input—or simply perform internal actions τ, or declare the experiment successful by generating the special action $\mathbf{W}$. Input states are instead those in which the experimenter is supposed to get some output generated by the system under test. They are always able to accept *any* value in L_U and, for each of such values, they proceed to the next output state.

As it clearly appears from the above description, an experimenter can be non-deterministic. The formal definition of experimenters follows:

Definition 4 (*Experimenter*). An *Experimenter* $\mathcal{T}$ over $L_I \times L_U$ is a tuple $(T_U, \upsilon_{\text{in}}, T_I, L_I, L_U, \rightarrow, \hookrightarrow)$ where T_U is the set of *output* states, υ_{in} is the *initial* state, T_I is the set of *input* states, $\rightarrow\ \subseteq (T_U \times L_I \times T_I) \cup (T_U \times \{\tau, \mathbf{W}\} \times T_U)$ is the *output transition relation*, and $\hookrightarrow \subseteq T_I \times L_U \times T_U$ is the *input transition relation*. The following three conditions must be satisfied: (i) υ_{in} is an output state, i.e. $\upsilon_{\text{in}} \in T_U$; (ii) $T_U \cap T_I = \emptyset$ and $(L_I \cup L_U) \cap \{\tau, \mathbf{W}\} = \emptyset$; (iii) $\hookrightarrow$ is *total*, i.e. $\forall \iota \in T_I, u \in L_U.\ \exists \upsilon \in T_U.\ (\iota, u, \upsilon) \in \hookrightarrow$.

We use similar shorthands as for LTSs, like $\upsilon \xrightarrow{i} \iota$ for $(\upsilon, i, \iota) \in \rightarrow$, $\iota \overset{u}{\hookrightarrow} \upsilon$ for $(\iota, u, \upsilon) \in \hookrightarrow$, etc. The *Success* set of the experimenter is the following set:

$$\{\upsilon \in T_U \mid \exists \upsilon' \in T_U.\ \upsilon \xrightarrow{\mathbf{W}} \upsilon'\}.$$

We say that an experimenter $\mathcal{T}$ is *finite* whenever T_U, T_I, $\rightarrow$, and $\hookrightarrow$ are finite sets. Similar considerations concerning finiteness as those for LTSs apply to experimenters as well.

Experimentation of a LTS over $L_I \times L_U$ against an experimenter $\mathcal{T}$ is modeled by the *Experimental System* they characterize:

Definition 5 (*Experimental system*). For LTS $\mathcal{M} = (S, s_{\text{in}}, L_I \times L_U, \rightarrow)$ and experimenter $\mathcal{T} = (T_U, \upsilon_{\text{in}}, T_I, L_I, L_U, \rightarrow, \hookrightarrow)$, the experimental system $\langle \mathcal{T}, \mathcal{M} \rangle$ is the transition system $(T_U \times S, (\upsilon_{\text{in}}, s_{\text{in}}), \rightsquigarrow)$. The transition relation $\rightsquigarrow \subseteq (T_U \times S) \times (T_U \times S)$ is the smallest relation satisfying the rules below where $s, s' \in S$, $i \in L_I$, $u \in L_U$, $\upsilon, \upsilon' \in T_U$, $\iota \in T_I$ and for $((\upsilon, s), (\upsilon', s')) \in \rightsquigarrow$ we write $\upsilon \mid\mid s \rightsquigarrow \upsilon' \mid\mid s'$:

$$\frac{\upsilon \xrightarrow{i} \iota,\ \iota \overset{u}{\hookrightarrow} \upsilon',\ s \xrightarrow{(i,u)} s'}{\upsilon \mid\mid s \rightsquigarrow \upsilon' \mid\mid s'} \qquad \frac{\upsilon \xrightarrow{\tau} \upsilon'}{\upsilon \mid\mid s \rightsquigarrow \upsilon' \mid\mid s}$$

Single experiments are modeled by *computations*:

Definition 6 (*Computations*). A *computation* of experimental system $\langle \mathcal{T}, \mathcal{M} \rangle$ is a sequence of the form:

$$\upsilon_0 \mid\mid s_0 \rightsquigarrow \upsilon_1 \mid\mid s_1 \rightsquigarrow \upsilon_2 \mid\mid s_2 \rightsquigarrow \ldots \upsilon_k \mid\mid s_k \rightsquigarrow \ldots$$

which is *maximal*, i.e. either it is infinite or it is finite with terminal element $\upsilon_n \mid\mid s_n$ which has the property that $\upsilon_n \mid\mid s_n \rightsquigarrow \upsilon' \mid\mid s'$ for no pair υ', s'. υ_0 and s_0 are the initial states υ_{in} and s_{in} of $\mathcal{T}$ and $\mathcal{M}$ respectively.

A computation is *successful* iff $\upsilon_k \in$ *Success* for some $k \geq 0$, otherwise it is *unsuccessful*.

We let $\text{Comp}(\mathcal{T}, \mathcal{M})$ denote the set of all computations of $\langle \mathcal{T}, \mathcal{M} \rangle$. From the definition of experimental system we know that every computation $\eta \in \text{Comp}(\mathcal{T}, \mathcal{M})$ gives rise to a transition $s_{\text{in}} \xrightarrow{\gamma}$ over finite or infinite sequence γ on the side of the LTS. In this case, we say that η *runs over* γ.

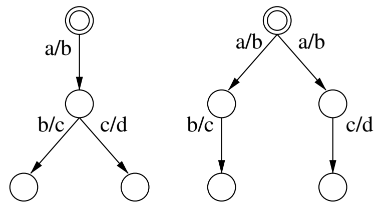


Fig. 3. Two Language-equivalent but not Testing equivalent LTSs.

Definition 7 (*Testing equivalence*). For experimental system $\langle \mathcal{T}, \mathcal{M} \rangle$ we let the set $\text{Result}(\mathcal{T}, \mathcal{M}) \subseteq \{\top, \bot\}$ be defined as follows:

$\top \in \text{Result}(\mathcal{T}, \mathcal{M})$ iff $\text{Comp}(\mathcal{T}, \mathcal{M})$ contains a successful computation.

$\bot \in \text{Result}(\mathcal{T}, \mathcal{M})$ iff $\text{Comp}(\mathcal{T}, \mathcal{M})$ contains an unsuccessful computation.

We say $\mathcal{M}$ and $\mathcal{M}'$ are *testing equivalent*, written $\mathcal{M} \sim \mathcal{M}'$ iff for all experimenters $\mathcal{T}$ $\text{Result}(\mathcal{T}, \mathcal{M}) = \text{Result}(\mathcal{T}, \mathcal{M}')$.

Intuitively, the above definition establishes that we can consider two systems $\mathcal{M}$ and $\mathcal{M}'$ equivalent if and only if no experimenter $\mathcal{T}$ can distinguish them on the basis of the fact that its computations have reported success or not. This notion completely captures the idea of equivalence based on the *externally observable* behavior of the two systems. Notice that testing equivalence for LTSs is strictly stronger than language equivalence for FSMs, in the sense that, as we shall see in Section 4, testing equivalent LTSs characterize the same language, while the converse does not hold. This can be easily seen using the example of Fig. 3. We leave it to the reader to prove that the two LTSs in the figure are not testing equivalent but they have the same language.

The following definition characterizes an interesting subclass of experimenters:

Definition 8 (*Input-deterministic experimenter*). Experimenter $\mathcal{T} = (T_U, \upsilon_{\text{in}}, T_I, L_I, L_U, \to, \hookrightarrow)$ is *input-deterministic* iff its input transition relation is deterministic, i.e. $\forall \iota \in T_I, u \in L_U, \upsilon, \upsilon' \in T_U.\ \iota \overset{u}{\hookrightarrow} \upsilon$ and $\iota \overset{u}{\hookrightarrow} \upsilon'$ implies $\upsilon = \upsilon'$.

In other words, each input state ι of input-deterministic experimenter $\mathcal{T}$ behaves like a total function from L_U to T_U. Of course input-deterministic experimenters can still exhibit non-deterministic behaviour since the output transition relation can be non-deterministic.

In the following we will let DetEx denote the class of all input-deterministic experimenters.

For set T of experimenters, we let $\mathcal{M} \sim^T \mathcal{M}'$ be the equivalence relativized to set T, i.e. $\mathcal{M} \sim^T \mathcal{M}'$ if and only if for all experimenters $\mathcal{T} \in T$, $\text{Result}(\mathcal{T}, \mathcal{M}) = \text{Result}(\mathcal{T}, \mathcal{M}')$.

The following proposition shows that for the purposes of reasoning about testing equivalence it is sufficient to consider input-deterministic experimenters.

Proposition 9. *For all LTSs $\mathcal{M}$ and $\mathcal{M}'$, $\mathcal{M} \sim \mathcal{M}'$ iff $\mathcal{M} \sim^{\mathsf{DetEx}} \mathcal{M}'$.*

In the sequel we will assume all experimenters be input-deterministic.

We close this section with the definition of a language for the specification of (input-deterministic) experimenters, which will be useful in the rest of the paper. Let IE and OS be countable sets such that $(IE \cup OS) \cap \{\tau, \mathbf{W}\} = \emptyset$—we call IE the set of *events* and

OS the set of *possible outputs*. The syntax of *output* experimenter expressions $\mathcal{U}$—resp. *input* experimenter expressions $\mathcal{I}$—of the language is given below, where $e \in IE$ is an event, $\alpha \in \{\tau, \mathbf{W}\}$, P (resp. x) is an experimenter (resp. input) variable, X is a parameter of type ΘD, g is a boolean expression of the form " $x = u$" or "$x \neq u$", for $u \in OS$, or "$x \notin X$" for $X \subseteq OS$. The notion of free (input) variable is the same as in lambda-calculus. The operators $+, \Rightarrow, ;$ and . have increasing binding strength. We let $\mathcal{U}, \mathcal{U}', \ldots$ range over *output* expressions, defined according to the following grammar:

$$\mathcal{U} ::= \delta \mid e; \mathcal{I} \mid \alpha; \mathcal{U} \mid g \Rightarrow \mathcal{U} \mid \mathcal{U} + \mathcal{U} \mid P(X)$$

Input expressions, ranged over by $\mathcal{I}, \mathcal{I}', \ldots$ are defined according to the following grammar:

$$\mathcal{I} ::= \lambda\, x.\mathcal{U}$$

An experimenter specification consists of a pair $(\mathcal{U}, U)$ where $\mathcal{U}$ is an output experimenter expression and $U \subseteq OS$. We will require that no input variable occurs free in $\mathcal{U}$ and that a unique experimenter definition $P(X) \stackrel{\Delta}{=} \mathcal{U}'$ is associated with any experimenter variable P occurring in $\mathcal{U}$ in the context where the experimenter specification is used. The operational semantics of the language is given in a similar way as for process algebraic languages, by means of a set of transition rules. The rules are shown in Fig. 4, with reference to experimenter specification $(\mathcal{U}, U)$ and a given set of process variable definitions. In the figure, μ is an element of $IE \cup \{\tau, \mathbf{W}\}$ and $\mathcal{O}$ stands both for output and for input experimenter expressions.

We briefly discuss the informal meaning of the rules. The experimenter δ performs no action. Prefix expression $e; \mathcal{I}$ offers event e and then behaves like $\mathcal{I}$, which is an input experimenter expression, i.e. an expression of the form $\lambda\, x.\mathcal{U}$. The latter will receive the output produced by the system under test in an experimental system (see Definition 5). The specific (output) state resulting from receiving the value is obtained according to the semantics of *input* experimenter expressions, as given by the last rule in Fig. 4 where $\mathcal{U}[u/x]$ denotes $\mathcal{U}$ where all free occurrences of x are simultaneously substituted with u. The second form of prefix expression, $\alpha; \mathcal{U}$, produces α and then behaves like $\mathcal{U}$. Notice that α can be either τ or the success action $\mathbf{W}$. In order for a conditional output experimenter $g \Rightarrow \mathcal{U}$ to proceed, it is necessary that the guard g evaluates to true. The choice expression $\mathcal{U}_1 + \mathcal{U}_2$ behaves as $\mathcal{U}_1$ or $\mathcal{U}_2$. Finally, if $P \stackrel{\Delta}{=} \mathcal{U}$ is the definition for P, P behaves like $\mathcal{U}$. If the optional parameter X is used in the definition of P, then $P(\mathcal{D})$ behaves as $\mathcal{U}[\mathcal{D}/X]$ where again we use substitution. In the following, we will assume $\mathcal{D}$ be an element of L, or L^*, 2^L, or Θ_D, the particular case being clear from the context. Finally, we will often use an extended form of parametrized experimenter $P(X_1, \ldots, X_k)$ with the obvious meaning. We say that an output experimenter expression $\mathcal{U}$ is *guarded* if and only if it is the body of an input experimenter expression $\lambda\, x.\mathcal{U}$, or it is the second argument of a prefix expression $\alpha; \mathcal{U}$ or of a conditional expression $g \Rightarrow \mathcal{U}$. In the sequel, we will consider only experimenter specifications where every occurrence of any (output) process instantiation $P(\mathcal{D})$ is guarded.

In order to formally define the experimenter denoted by an experimenter specification we first need the following auxiliary definition where by $\vdash \mathcal{U} \stackrel{\mu}{\longrightarrow} \mathcal{O}$ (resp. $\vdash \mathcal{I} \stackrel{u}{\hookrightarrow} \mathcal{U}$) we mean that $\mathcal{U} \stackrel{\mu}{\longrightarrow} \mathcal{O}$ (resp. $\mathcal{I} \stackrel{u}{\hookrightarrow} \mathcal{U}$) can be deduced using the rules of Fig. 4.

Definition 10 (*Derivatives*). The *derivatives* of experimenter specification $(\mathcal{U}, U)$ is the smallest set $\mathsf{der}_{(\mathcal{U},\, U)}$ of experimenter expressions which satisfies the following three conditions: (i) $\mathcal{U} \in \mathsf{der}_{(\mathcal{U},\, U)}$; (ii) if output experimenter expression $\mathcal{U}' \in \mathsf{der}_{(\mathcal{U},\, U)}$ and

$$e;\mathcal{I} \xrightarrow{e} \mathcal{I} \qquad \alpha;\mathcal{U} \xrightarrow{\alpha} \mathcal{U}$$

$$\frac{\mathcal{U} \xrightarrow{\mu} \mathcal{O}}{\mathcal{U} + \mathcal{U}' \xrightarrow{\mu} \mathcal{O}} \qquad \frac{\mathcal{U} \xrightarrow{\mu} \mathcal{O}}{\mathcal{U}' + \mathcal{U} \xrightarrow{\mu} \mathcal{O}}$$

$$\frac{\mathcal{U} \xrightarrow{\mu} \mathcal{O}}{TRUE \Rightarrow \mathcal{U} \xrightarrow{\mu} \mathcal{O}} \qquad \frac{P(X) \triangleq \mathcal{U}, \mathcal{U}[\mathcal{D}/X] \xrightarrow{\mu} \mathcal{O}}{P(\mathcal{D}) \xrightarrow{\mu} \mathcal{O}}$$

$$\frac{u \in U}{\lambda\, x.\mathcal{U} \overset{u}{\hookrightarrow} \mathcal{U}[u/x]}$$

Fig. 4. Experimenter expressions operational semantics rules.

$\vdash \mathcal{U}' \xrightarrow{\mu} \mathcal{O}$ then also $\mathcal{O} \in \mathsf{der}_{(\mathcal{U},\, U)}$; (iii) if input experimenter expression $\mathcal{I} \in \mathsf{der}_{(\mathcal{U},\, U)}$ and $\vdash \mathcal{I} \overset{u}{\hookrightarrow} \mathcal{U}'$ for some $u \in U$, then also $\mathcal{U}' \in \mathsf{der}_{(\mathcal{U},\, U)}$.

We can now define the experimenter associated with experimenter specification $(\mathcal{U}, U)$:

Definition 11 (*Operational semantics of Experimenters*). The experimenter associated with experimenter specification $(\mathcal{U}, U)$ is the experimenter $(T_U, \upsilon_{\text{in}}, T_I, L_I, L_U, \rightarrow, \hookrightarrow)$ where $T_U =_{df} \{\mathcal{U}' \mid \mathcal{U}' \in \mathsf{der}_{(\mathcal{U},U)}\}$, $\upsilon_{\text{in}} = \mathcal{U}$, $T_I =_{df} \{\mathcal{I} \mid \mathcal{I} \in \mathsf{der}_{(\mathcal{U},U)}\}$, $L_I \subseteq IE$, $L_U =_{df} U$, and $\rightarrow, \hookrightarrow$ are the smallest relations induced by the rules of Fig. 4.

In the sequel we will omit set U in experimenter specification $(\mathcal{U}, U)$ when U is clear from the context. Moreover we will identify $(\mathcal{U}, U)$ with the experimenter it denotes. The following is an example of a very simple experimenter over $I \times U$, where $I = \{r_1\}$ and $U = \{\{a_1\}, \{e_1\}, \{r_2\}\}$ which starts by sending r_1 to the system under test and then, if the latter responds with $\{a_1\}$ it reports success, otherwise it stops without reporting success:

$$\begin{array}{ll} r_1;\, \lambda x.\, & (x = \{a_1\} \Rightarrow \tau;\, \mathbf{W};\, \delta \\ & + \\ & x \notin \{\{a_1\}\} \Rightarrow \delta \\ &) \end{array}$$

4. Testing relations

In this section we develop a general testing theory for UMLSCs, originally proposed in [25], using a framework similar to that proposed in [30], which was in turn inspired by the work of Hennessy for traditional LTSs [17]. The general approach is based on the well known notions of *MAY* and *MUST* preorders and related equivalences. The main semantic assumptions in [17] are that (i) system interaction is modeled by action-synchronization rather than input/output exchanges, and (ii) absence of reaction from a system to a stimulus presented by an experimenter results in a deadlock affecting both the system and the experimenter. In [30], and later in [25] specifically for UMLSCs, assumption (i) has been replaced by modeling system interaction as input/output exchanges, but assumption (ii) remains unchanged. In the following we shall first recall the semantic interpretation of HAs as proposed in [25], which we call the *non-stuttering* semantics for HAs, for reasons which will be clear in the sequel, and we show its formal relation with the original semantics for HAs recalled in Section 3.2. In the rest of this section we will develop the above-mentioned testing theory based on the non-stuttering semantics.

The non-stuttering semantics is recalled in Section 4.1 while its relationship with the original semantics of UMLSCs proposed in [15] is addressed in Section 4.2. In Section 4.3 relevant testing preorders are given which brings us to the notion of testing equivalence. In Section 4.4 an alternative, intensional, characterization of such preorders/equivalence is addressed which serves as a link to a finite representation for (the LTSs denoted by) UMLSCs used for automatic verification in Section 4.6.

4.1. Non-stuttering semantics

The non-stuttering semantics associates a LTS to each HA.

Definition 12 (*Non-stuttering semantics*). The non-stuttering semantics of an HA $H = (F, E, \rho)$ is the LTS over $E \times \Theta_E$ $\mathsf{LTS}(H) =_{df} (\mathrm{Conf}_H, \mathcal{C}_{\mathrm{in}}, \longrightarrow)$ where (i) Conf_H is the set of configurations, (ii) $\mathcal{C}_{\mathrm{in}} \in \mathrm{Conf}_H$ is the initial configuration, (iii) $\longrightarrow \subseteq \mathrm{Conf}_H \times (E \times \Theta_E) \times \mathrm{Conf}_H$ is the *step*-transition relation defined below.

We write $\mathcal{C} \xrightarrow{e/\mathcal{E}} \mathcal{C}'$ for $(\mathcal{C}, (e, \mathcal{E}), \mathcal{C}') \in \longrightarrow$. Any such transition denotes the result of firing a maximal set of non-conflicting transitions of the sequential automata of H which respect priorities when the state machine associated to H is given event e as an input. $\mathcal{E}$ is the collection of output events generated by the transitions which have been fired. Relation $\longrightarrow$ is the smallest relation which satisfies the rule below:

Definition 13 (*Non-stuttering semantics transition rule*).

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

Also in the above rule, as in Definition 3, we make use of the Core Semantics. It is worth pointing out that the LTS associated by non-stuttering semantics to a generic HA is *finite*. This nice property can be easily understood by considering that each HA has a finite set of events, a finite set of configurations and that the total set of transitions is finite, so that there is a finite number of subsets of transitions, i.e. there is a finite number of possible step-transitions. For HA H as in Fig. 2, the corresponding $\mathsf{LTS}(H)$ is shown in Fig. 5.[4]

[4] For the sake of simplicity, in the examples in the present paper, the events generated as outputs are collected as *sets*, i.e. Θ_E is chosen to be 2^E; moreover, a singleton set $\{e\}$ is denoted by the element e it contains, when this cannot cause confusion.

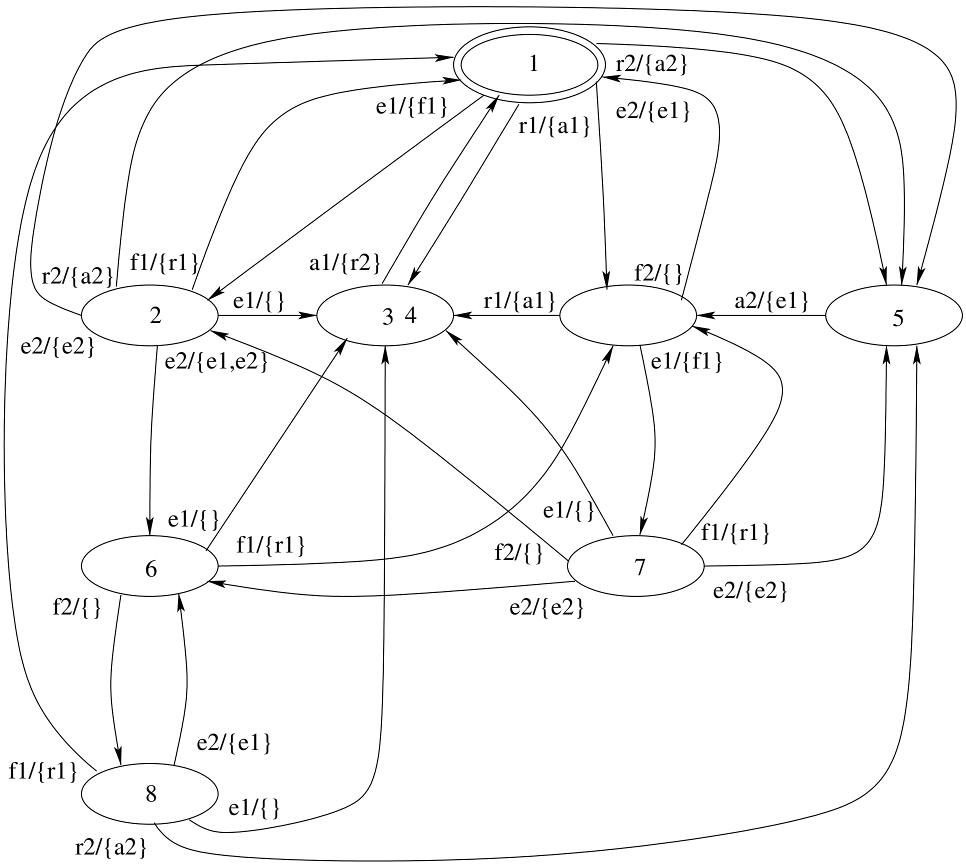


Fig. 5. LTS(H), for H of Fig. 2.

4.2. Correctness

The original operational semantics proposed in [24] was proved correct w.r.t. the official UML Statechart Diagrams semantics, although the latter are defined only informally. The correctness theorem in [24] essentially states that the set of transitions fired during an arbitrary step is a *maximal* set $\mathcal{L}$ such that (a) all transitions in $\mathcal{L}$ are enabled, (b) they are non-conflicting and (c) there is no transition outside $\mathcal{L}$ which is enabled in the current status and which has higher priority than a transition in $\mathcal{L}$.

In this section we shall provide a correctness result for the non-stuttering semantics, showing its formal relation with the original semantics, in the form presented in [15] and recalled in Section 3.2 (Definition 3), which differs from that presented in [24] only in that each step-transition is explicitly labeled by set $\mathcal{L}$, which is omitted in [24].

In order to present the correctness result it is convenient to model the UML input-queue as a specific experimenter, namely the experimenter which simulates the data structure used for the queue. The experimenter we are interested in is specified by $(Queue(\mathcal{E}_0), \Theta_E)$ where $\mathcal{E}_0$ is the initial content of the input-queue and $Queue$ is recursively defined, as shown in Fig. 6.

$$Queue(\mathcal{F} : \Theta_E) \triangleq$$
$$(\mathsf{Sel}\ \mathcal{F}\ f\ \mathcal{F}') \Rightarrow (f; \lambda\mathcal{F}''.Queue(\mathsf{Join}\ \mathcal{F}'\ \mathcal{F}''))$$

Fig. 6. Definition of $Queue(\mathcal{F})$.

Intuitively, for structure $\mathcal{E}$ in Θ_E, $Queue(\mathcal{E})$ will produce a transition if and only if there exist event e and structure $\mathcal{E}'$ such that $(\mathsf{Sel}\ \mathcal{E}\ e\ \mathcal{E}')$ holds, i.e. $\mathcal{E}$ is not empty. Moreover, f is bound to e and $\mathcal{F}'$ is bound to $\mathcal{E}'$. The expression $(f; \lambda\mathcal{F}''.Queue(join\ \mathcal{F}'\ \mathcal{F}''))$ is an *action prefix* which performs the event (action) currently bound to f, say e, and then behaves like $\lambda\mathcal{F}''.Queue(join\ \mathcal{F}'\ \mathcal{F}'')$. The latter, when receiving a structure, say $\mathcal{E}''$, will behave again like a queue but with a different argument, i.e. $Queue(join\ \mathcal{E}'\ \mathcal{E}'')$.

The following proposition, proved in Appendix B, establishes the correctness of the new semantics definition. The transition relation in the operational semantics given in [15] is denoted by $\xrightarrow{\mathcal{L}}$.

Proposition 14. *For HA* $H = (F, E, \rho), \mathcal{C}, \mathcal{C}' \in \mathrm{Conf}_H, \mathcal{E}, \mathcal{E}', \mathcal{E}'' \in \Theta_E$, *the following holds*: $\exists\mathcal{L}.\ \mathcal{L} \neq \emptyset\ \wedge\ (\mathcal{C}, \mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}', (join\ \mathcal{E}''\ \mathcal{E}'))$ *if and only if* $(Queue(\mathcal{E})\ ||\ \mathcal{C}) \rightsquigarrow (Queue(join\ \mathcal{E}''\ \mathcal{E}')\ ||\ \mathcal{C}')$.

So the two semantic models 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 modeled in the new semantics by not generating a step-transition at all. This last behavior is in line with traditional testing theories as developed e.g. in [17].

4.3. Testing preorders

Below we define preorders which will allow us to "order" non-deterministic i/o-pair-LTSs, like HAs, according to their "amount of non-determinism" and to recollect testing equivalence as the equivalence induced by such preorders.

Definition 15 (*Testing preorders*). For $\mathcal{M}$, $\mathcal{M}'$ i/o-pair LTSs we let

(i) $\mathcal{M} \sqsubseteq_{MAY} \mathcal{M}'$ iff for every $\mathcal{T}$: if $\top \in \mathrm{Result}(\mathcal{T}, \mathcal{M})$ then also $\top \in \mathrm{Result}(\mathcal{T}, \mathcal{M}')$;
(ii) $\mathcal{M} \sqsubseteq_{MUST} \mathcal{M}'$ iff for every $\mathcal{T}$: if $\perp \notin \mathrm{Result}(\mathcal{T}, \mathcal{M})$ then also $\perp \notin \mathrm{Result}(\mathcal{T}, \mathcal{M}')$;
(iii) $\mathcal{M} \sqsubseteq \mathcal{M}'$ iff $(\mathcal{M} \sqsubseteq_{MAY} \mathcal{M}') \wedge (\mathcal{M} \sqsubseteq_{MUST} \mathcal{M}')$.

So $\mathcal{M} \sqsubseteq_{MAY} \mathcal{M}'$ means that if a generic experimenter $\mathcal{T}$ *may* report success when experimenting with $\mathcal{M}$ it must be the case that $\mathcal{T}$ *may* report success also when experimenting with $\mathcal{M}'$. Symmetrically, $\mathcal{M} \sqsubseteq_{MUST} \mathcal{M}'$ means that if a generic experimenter $\mathcal{T}$ *must* report success when experimenting with $\mathcal{M}$ it must be the case that $\mathcal{T}$ *must* report success also when experimenting with $\mathcal{M}'$. In other words if we know that $\mathcal{M}$ *may* pass a generic test $\mathcal{T}$ and $\mathcal{M} \sqsubseteq_{MAY} \mathcal{M}'$ then we know also that $\mathcal{M}'$ *may* pass the test, where "*may* pass the test" is the informal equivalent of $\top \in \mathrm{Result}(\mathcal{T}, \mathcal{M})$, with the intuitive meaning that there may be a successful computation starting from the initial state of the experimental system

$\langle \mathcal{T}, \mathcal{M} \rangle$. Similarly if we know that $\mathcal{M}$ *must* pass a generic test $\mathcal{T}$ and $\mathcal{M} \sqsubseteq_{MUST} \mathcal{M}'$ then we know also that $\mathcal{M}'$ *must* pass the test, where "*must* pass the test" is the informal equivalent of $\perp \notin \text{Result}(\mathcal{T}, \mathcal{M})$, with the intuitive meaning that all computations starting from the initial state of the experimental system $\langle \mathcal{T}, \mathcal{M} \rangle$ must be successful.

We let $\eqsim$ denote the equivalence induced by the testing preorders, i.e. $\mathcal{M} \eqsim \mathcal{M}'$ iff $\mathcal{M} \sqsubseteq \mathcal{M}'$ and $\mathcal{M}' \sqsubseteq \mathcal{M}$. The proposition below allows to identify $\sim$ with $\eqsim$.

Proposition 16. $\mathcal{M} \sim \mathcal{M}'$ *iff* $\mathcal{M} \eqsim \mathcal{M}'$.

Finally, we let $\sqsubseteq^{\aleph}_{MAY}$, $\sqsubseteq^{\aleph}_{MUST}$, $\sqsubseteq^{\aleph}$ denote the relativized preorders. For example $\mathcal{M} \sqsubseteq^{\aleph}_{MUST} \mathcal{M}'$ iff for every $\mathcal{T} \in \aleph$, $\perp \notin \text{Result}(\mathcal{T}, \mathcal{M}) \Rightarrow \perp \notin \text{Result}(\mathcal{T}, \mathcal{M}')$.

4.4. Alternative characterization of testing preorders

The characterization of testing preorders by means of the concepts of *MAY* and *MUST* is intuitively appealing, but is problematic when it comes to automatic verification of testing equivalence of LTSs. Such an automatic verification can be performed based on so called *Acceptance Automata*, which are a variant of Acceptance Trees originally proposed by Hennessy. In order to be able to show the correspondence between testing equivalence defined in terms of the *MAY* and *MUST* preorders and the *Acceptance Automata*—which will be introduced in the next section—we give an intermediate alternative characterization of testing preorders in this section. First we introduce two auxiliary notions; *set closure* and *maximal functional subsets*.

Definition 17 (*Set closure*). For X a finite set of finite subsets of L, the *closure* of X, $\mathbf{c}\ X$ is the smallest set such that the following three conditions are satisfied:
(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 $x_1 \subseteq x \subseteq x_2$ then also $x \in \mathbf{c}\ X$.

The following definition is necessary for identifying the *functional* subsets of subsets of L whenever L is a set of i/o-pairs. Functional sets, which are in fact (finite) functions, are used for modeling single steps of input/output behavior.

Definition 18 (*mfs*). For X a finite set of finite subsets of L we let

$$mfs\ X =_{df} \bigcup_{x \in X} (mf\ x)$$

where $mf\ x =_{df} \{y \in (func\ x) \mid \nexists y' \in (func\ x).\ y \subset y'\}$ and $func\ x =_{df} \{y \subseteq x \mid \forall (i_1, u_1), (i_2, u_2) \in y. i_1 = i_2 \Rightarrow u_1 = u_2\}$.

For finite set X of finite subsets of L, $mfs\ X$ generates the *maximal functional subsets* of the elements of X, by applying function mf to each of them. Function mf splits each set into its maximal functional subsets. Each functional set is indeed a *function* from input-events to output-events. As we will see, intuitively, every such a set represents an instance of *external* non-determinism relative to a single step of the machine. Similarly, but in a complementary way, *internal* non-determinism relative to a single step of the machine is coded by means of having different functional sets as elements of the same set associated to such a step. Notice that $func\ \emptyset = \{\emptyset\} = mf\ \emptyset$ and $mfs\ \{\emptyset\} = \{\emptyset\}$. The following definitions introduce a preorder on finite LTSs which will be used for defining the intermediate equivalence $\approx$ and which will be proved to coincide with the testing preorder.

Definition 19 (*Alternative preorders*). For finite LTSs $\mathcal{M}, \mathcal{M}'$ over $L = L_I \times L_U$

(i) $\mathcal{M} \ll_{MAY} \mathcal{M}'$ iff $(\mathsf{lan}\ \mathcal{M}) \subseteq (\mathsf{lan}\ \mathcal{M}')$;

(ii) $\mathcal{M} \ll_{MUST} \mathcal{M}'$ iff $\forall \gamma \in L^*.\ mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma)) \subset\subset mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$;

(iii) $\mathcal{M} \ll \mathcal{M}'$ iff $\mathcal{M} \ll_{MAY} \mathcal{M}' \wedge \mathcal{M} \ll_{MUST} \mathcal{M}'$;

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

It is easy to show that $\ll$ is indeed a preorder so that it induces the following equivalence:

Definition 20 (*Alternative equivalence*). For finite LTSs $\mathcal{M}, \mathcal{M}'$ over L $\mathcal{M} \approx \mathcal{M}'$ iff $\mathcal{M} \ll \mathcal{M}' \wedge \mathcal{M}' \ll \mathcal{M}$.

The following theorem establishes the first correspondence result, namely the correspondence between the testing preorders (Definition 15) and the preorders defined in Definition 19:

Theorem 21. *For all finite LTSs $\mathcal{M} = (S, s_{in}, L, \rightarrow)$, $\mathcal{M}' = (S', s'_{in}, L, \rightarrow)$ over $L = L_I \times L_U$ the following holds*:

(a) $\mathcal{M} \sqsubseteq\!\!\!\!\sim_{MAY} \mathcal{M}'$ *iff* $\mathcal{M} \ll_{MAY} \mathcal{M}'$;

(b) $\mathcal{M} \sqsubseteq\!\!\!\!\sim_{MUST} \mathcal{M}'$ *iff* $\mathcal{M} \ll_{MUST} \mathcal{M}'$;

(c) $\mathcal{M} \sqsubseteq\!\!\!\!\sim \mathcal{M}'$ *iff* $\mathcal{M} \ll \mathcal{M}'$.

As a corollary we have the link between testing equivalence and the relation $\approx$ defined on LTSs.

Corollary 22. *For finite LTSs $\mathcal{M}, \mathcal{M}'$ over $L = L_I \times L_U$, $\mathcal{M} \eqsim \mathcal{M}'$ iff $\mathcal{M} \approx \mathcal{M}'$.*

At this point we can already say something about the exact nature of this notion of testing LTSs over L. Essentially it has to do with internal non-determinism, as it can be detected by means of "black-box" testing. Intuitively $\mathcal{M} \sqsubseteq\!\!\!\!\sim \mathcal{M}'$ if they have the same set of traces but in some sense $\mathcal{M}$ is "more non-deterministic", or equivalently, "more chaotic" then $\mathcal{M}'$. In other words, although the sequences of input/output interactions of both systems are the same, an experimenter $\mathcal{T}$ may experience failures with $\mathcal{M}$ "more often" than with $\mathcal{M}'$. In this sense, $\mathcal{M}$ has a "higher degree of non-determinism" than $\mathcal{M}'$.

4.5. Finite acceptance automata

In this section we introduce the model of *finite acceptance automata* (FAAs), equipped with a preorder $\leq_{FAA}$ and the equivalence relation $\equiv_{FAA}$ it induces. FAAs are a natural extension of Finite Acceptance Trees to the case of i/o-pairs LTS. Finite Acceptance Trees have been originally introduced by Hennessy [17]; they have been adapted to the case of systems with explicit input/output behavior in [30]. Both in [17,30] Acceptance Trees form a semantic domain within a denotational approach. So, Acceptance Trees modeling systems with traces of unlimited length—even finite state systems—are characterized by infinite fixpoints. In this paper, instead, we are using an operational—rather than denotational—approach and we shall use FAAs for modeling the behavior of finite state systems with i/o-labels, including those with traces of unlimited length. In other words, our acceptance structures are generic finite graphs and not simply finite trees or directed acyclic graphs.

The reason why we introduce FAAs is fairly simple: they can easily be mapped into finite *deterministic* LTSs on which strong bisimulation equivalence can be automatically checked, and such a mapping preserves the equivalence $\equiv_{FAA}$ on FAAs. On the other hand, we can map finite LTSs over i/o-pairs into FAAs in such a way that *testing equivalence* is preserved, i.e. two finite LTSs over i/o-pairs are testing equivalent if and only if their images via such a mapping are equivalent according to $\equiv_{FAA}$. In conclusion, FAAs represent an effective model for performing automatic verification of testing equivalence over UMLSCs. Before defining the FAA model, we need to define the notion of *saturated sets*.

Definition 23 (*Saturated sets*). For finite subset S of L, an S-set A is a finite, non-empty set of subsets of L which satisfies the following conditions: (i) $\forall X \in A.\ X \subseteq S$; (ii) $\forall x \in S.\ \exists X \in A.\ x \in X$; (iii) $\forall X_1, X_2 \in A.\ X_1 \cup X_2 \in A$; and (iv) $\forall X_1, X_2 \in A, X.\ X_1 \subseteq X \subseteq X_2 \Rightarrow X \in A$. A finite set A of finite subsets of L is *saturated* if it is an S-set for some S.

Finite acceptance automata (FAAs) are defined below:

Definition 24 (*FAA*). A *finite acceptance automaton* α over L is a deterministic finite LTS over L, where also nodes are labeled.[5] The node of α identified by sequence $\gamma \in \mathsf{lan}\ \alpha$ is labeled by $(mfs\ A)$ for some $(\mathsf{S}\ \alpha\ \gamma)$-set A. Such a label will be denoted by $\mathsf{AS}_{\mathrm{FAA}}\ \alpha\ \gamma$, and is assumed equal to $\emptyset$ whenever $\gamma \notin \mathsf{lan}\ \alpha$.

It is easy to see that the relation on FAA defined below is a preorder (but not a partial order).

Definition 25 ($\leq_{FAA}$ *and* $\equiv_{FAA}$). For FAAs α, α' over L, $\alpha \leq_{FAA} \alpha'$ iff the following conditions are satisfied: (i) $\mathsf{lan}\ \alpha = \mathsf{lan}\ \alpha'$, and (ii) $\forall \gamma \in \mathsf{lan}\ \alpha.\ \mathsf{AS}_{\mathrm{FAA}}\ \alpha'\ \gamma \subseteq \mathsf{AS}_{\mathrm{FAA}}\ \alpha\ \gamma$. The equivalence induced by $\leq_{FAA}$ is denoted by $\equiv_{FAA}$.

Intuitively, $\alpha \leq_{FAA} \alpha'$ if they have the same set of traces but α represents "more non-deterministic" systems. Such non-determinism is coded in the acceptance sets. In order to compute the FAA $(\mathsf{T}_{\mathsf{FAA}}\ \mathcal{M})$ associated to any $\mathcal{M}$, finite LTS over L, we proceed in a similar way as in [4]. The algorithm is defined in Fig. 7.

Proposition 26. *For finite LTS $\mathcal{M}$ over $L = L_I \times L_U$, $\mathsf{T}_{\mathsf{FAA}}\ \mathcal{M}$ is a FAA over L.*

Fig. 9(a) shows the result of applying mapping $\mathsf{T}_{\mathsf{FAA}}$ to the LTS of Fig. 8(b) which is the semantics of the UMLSC of Fig. 8(a). The states of the UMLSC have been numbered for notational convenience; the relevant sets of such numbers are used as names of the states of of the LTS and and the FAA.

5 All definitions for LTS are thus valid also for FAAs.

For $\mathcal{M}$ finite LTS over $L = L_I \times L_U$, let $(\mathsf{T}_{\mathsf{FAA}}\ \mathcal{M})$ be the FAA computed as follows:

(1) Apply the Aho-Ullman "Subset Construction" Algorithm ([1], pag. 93) to $\mathcal{M}$, getting deterministic automaton $d = (D, d_{\text{in}}, \rightarrow_d)$ over L;

(2) Let $(\mathsf{T}_{\mathsf{FAA}}\ \mathcal{M})$ be the FAA $\alpha = (A, \alpha_{\text{in}}, \rightarrow_\alpha, \mathsf{AS}_{\text{FAA}}\ \alpha)$ defined as follows:

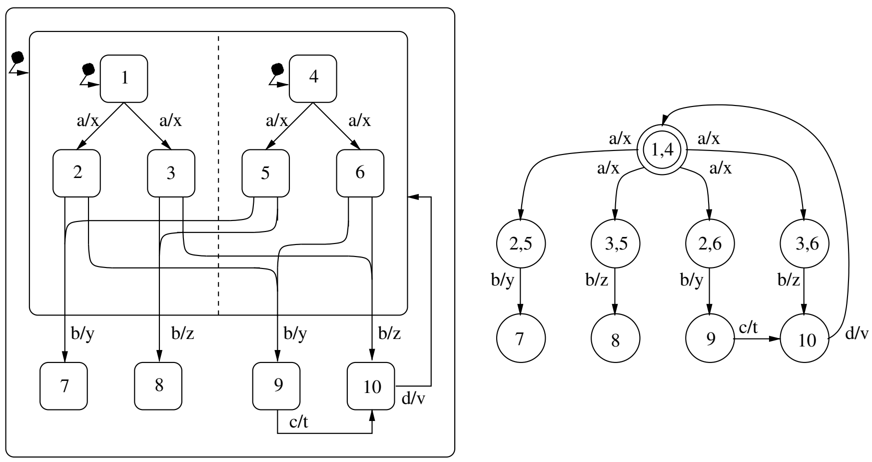
  (a) $A = D$;

  (b) $\alpha_{\text{in}} = d_{\text{in}}$;

  (c) For all $\alpha', \alpha'' \in A, (i,u) \in L$, $\alpha' \xrightarrow{(i,u)}_\alpha \alpha''$ iff $\alpha' \xrightarrow{(i,u)}_d \alpha''$;

  (d) For all $\gamma \in L^*$ let
  - $\mathsf{AS}_{\text{FAA}}\ \alpha\ \gamma = mfs\ (\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$, if $\gamma \in \mathsf{lan}\ \mathcal{M}$;
  - $\mathsf{AS}_{\text{FAA}}\ \alpha\ \gamma = \emptyset$, if $\gamma \notin \mathsf{lan}\ \mathcal{M}$.

Fig. 7. The algorithm for mapping $\mathsf{T}_{\mathsf{FAA}}$.



(a) **(b)**

Fig. 8. A simple UMLSC (a) and its LTS (b).

We are now ready for giving the second correspondence theorem.

Theorem 27. *For all finite LTSs* $\mathcal{M}, \mathcal{M}'$ *over* $L = L_I \times L_U$ *the following holds*: $\mathcal{M} << \mathcal{M}'$ *iff* $(T_{FAA}\ \mathcal{M}) \leq_{FAA} (T_{FAA}\ \mathcal{M}')$.

As a corollary of Theorem 27 we have the link between the relation $\approx$ on LTSs and the equivalence of FAAs.

Corollary 28. *For finite LTSs* $\mathcal{M}, \mathcal{M}'$ *over* L, $\mathcal{M} \approx \mathcal{M}'$ *iff* $(T_{FAA}\ \mathcal{M}) \equiv_{FAA} (T_{FAA}\ \mathcal{M}')$.

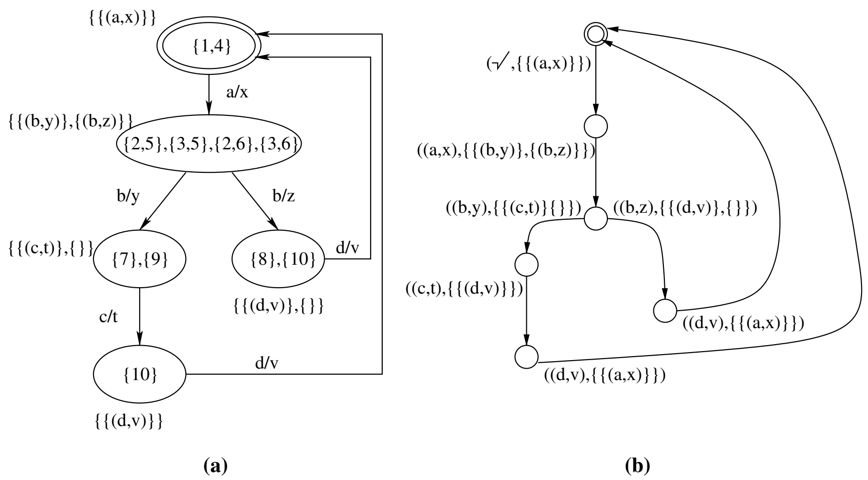


Fig. 9. FAA (a) and related deterministic LTS (b) associated to the LTS of Fig. 8.

4.6. Testing equivalence verification

In this section we show how the results given in the previous section can be used for effective verification of testing equivalence over UMLSCs. We start by stating the final correspondence result, which easily follows from Corollaries 22 and 28.

Corollary 29. *For all finite LTSs $\mathcal{M}, \mathcal{M}'$ over L, $\mathcal{M} \eqsim \mathcal{M}'$ iff* $(\mathsf{T_{FAA}}\ \mathcal{M}) \equiv_{FAA} (\mathsf{T_{FAA}}\ \mathcal{M}')$.

The above result allows us to reduce the problem of checking whether two UMLSCs S and S' are testing equivalent to the problem of checking whether $(\mathsf{T_{FAA}}\ (\mathsf{LTS}(H))) \equiv_{FAA} (\mathsf{T_{FAA}}\ (\mathsf{LTS}(H')))$ where H (resp. H') is the HA representing S (resp. S'). In the remainder of the paper we show that checking $\alpha \equiv_{FAA} \alpha'$, for FAAs α, α', can be reduced in turn to checking (strong) bisimulation equivalence. Below we recall the definition of bisimulation equivalence [33].

Definition 30 (*Bisimulation equivalence*). A binary relation R on states of LTSs over label set L is a (strong) bisimulation iff for all $l \in L$ and s_1, s_2 with $s_1\ R\ s_2$

- whenever $s_1 \xrightarrow{l} s_1'$ for some s_1' also $s_2 \xrightarrow{l} s_2'$ for some s_2' such that $s_1'\ R\ s_2'$, and
- whenever $s_2 \xrightarrow{l} s_2'$ for some s_2' also $s_1 \xrightarrow{l} s_1'$ for some s_1' such that $s_1'\ R\ s_2'$.

We say that s_1 and s_2 are (strong) bisimulation equivalent, written $s_1 \approx_{bis} s_2$ in this paper, iff there exists a bisimulation R such that $s_1\ R\ s_2$.

Two LTSs $\mathcal{M}$ and $\mathcal{M}'$ are bisimulation equivalent, written $\mathcal{M} \approx_{bis} \mathcal{M}'$, if and only if their initial states are so. It is important to point out that there are tools available nowadays for automatic verification of bisimulation equivalence for finite LTSs (see, e.g. [13]). In order to reduce our problem to bisimulation equivalence checking we first build the LTS $(\mathsf{Up}\ \alpha)$ for FAA α according to the algorithm shown in Fig. 10. The algorithm simply moves node labels up to the transitions pointing to such nodes, introducing a new node and

For FAA over L $\alpha = (A, \alpha_{\text{in}}, \rightarrow_\alpha, \mathsf{AS}_{\text{FAA}}\ \alpha)$, let $\mathsf{Up}\ \alpha$ be the finite LTS over $L \times 2^{2^L}$, $d = (D, d_{\text{in}}, \rightarrow_d)$ defined as follows:
(1) $D = A \cup \{d_{\text{in}}\}$, with $d_{\text{in}} \notin A$;
(2) $\rightarrow_d$ is the smallest relation over $D \times (L \times 2^{2^L}) \times D$ such that the following holds:
- $d_{\text{in}} \xrightarrow{v}_d \alpha_{\text{in}}$ with $v = (\surd, \mathsf{AS}_{\text{FAA}}\ \alpha\ \epsilon)$ and $\surd \notin L$;
- For all $d', d'' \in D$, $x \in L$ and $\gamma \in L^*$ such that γ identifies d' in α and γx identifies d'' in α, $d' \xrightarrow{v}_d d''$ iff $d' \xrightarrow{x}_\alpha d''$ and $v = (x, \mathsf{AS}_{\text{FAA}}\ \alpha\ \gamma x)$.

Fig. 10. The algorithm for mapping Up.

a new transition for the label of the initial node. Fig. 9(b) shows the result of applying the algorithm to the FAA of Fig. 9(a). It is easy to see that the two lemmas below directly follow from the definitions of Up and $\equiv_{FAA}$.

Lemma 31. *For FAA α over L, $(\mathsf{Up}\ \alpha)$ is a deterministic, finite LTS.*

Lemma 32. *For FAAs α and α' over L the following holds: $\alpha \equiv_{FAA} \alpha'$ iff $\mathsf{lan}\ (\mathsf{Up}\ \alpha) = \mathsf{lan}\ (\mathsf{Up}\ \alpha')$.*

But then, since strong bisimulation equivalence coincides with trace equivalence for deterministic LTSs (see e.g. [22]), from the above two lemmas we can easily prove the following:

Theorem 33. *For all finite LTSs $\mathcal{M}, \mathcal{M}'$ over $L = L_I \times L_U$ the following holds: $\mathcal{M} \eqsim \mathcal{M}'$ if and only if $\mathsf{Up}(\mathsf{T}_{\mathsf{FAA}}\ \mathcal{M}) \approx_{bis} \mathsf{Up}(\mathsf{T}_{\mathsf{FAA}}\ \mathcal{M}')$.*

We leave it to the reader to verify that the UMLSC of Fig. 11 is testing equivalent[6] to that of Fig. 8(a).

5. Conformance testing

Broadly speaking, 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) at hand. An account of the major results in the area of conformance relations and conformance testing can be found in [42]. The theory has been developed mainly in the context of process algebras and input/output LTSs. Input/output LTSs[7] [29] are LTSs where the set of labels is *partitioned* into two separate sets, i.e. input labels and output ones. Moreover, in the context of conformance testing theories, such LTSs are required to be *input enabled*, i.e. for each label of the input set, in each state of the LTS there must be at least one outgoing transition labeled by such a label. Finally, the situation in which, in a

[6] Actually, the associated deterministic LTSs turn out to be not only bisimulation equivalent, but also isomorphic. Obviously this does not need to be the case in general.
[7] Strictly speaking "Automata".

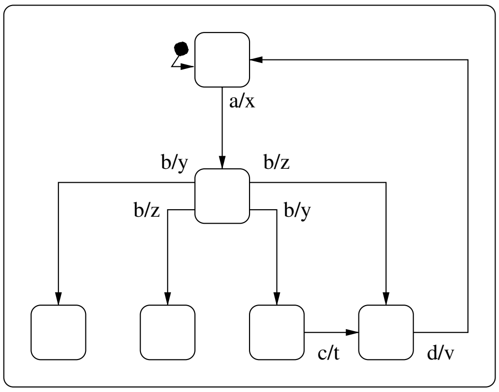


Fig. 11. A UMLSC testing-equivalent to that of Fig. 8.

given state, an LTS does not generate any output at all is modeled by an outgoing transition labeled by a special label denoting "quiescence". Both specifications and implementations are modeled by input enabled LTSs with quiescence.

Under the above modeling assumptions, one of the most successful formal conformance relations is the **ioco** relation proposed by Tretmans in the above-mentioned work [42]. Informally, for specification $\mathcal{M}$ and implementation $\mathcal{M}'$, $\mathcal{M}'$ **ioco** $\mathcal{M}$ means that $\mathcal{M}'$ can never produce an output which could not be produced by $\mathcal{M}$ "in the same situation", i.e. after the same sequence of steps.

In the previous section we have developed a model of system behavior based on the assumption that absence of reaction by a system to a stimulus presented by an experimenter results in a deadlock affecting both the system and the experimenter. In this section, instead, we define a slightly richer semantics for HAs, which we call the *stuttering semantics*, where absence of reaction is represented *explicitly* in the associated LTSs, in a way which is similar to quiescence and which naturally represents the notion of *stuttering* in the context of UMLSCs. A HA H (or equivalently the UMLSC it represents) experiences a stuttering step on input event e whenever, in the current configuration $\mathcal{C}$ *no* transition is enabled on such input e. The input event e is consumed anyway but no state change occurs in H.[8] As we will see, in the stuttering semantics when stuttering occurs, the output component of the label of the involved step-transition is the special symbol Σ. Thus, in the remainder of this paper we will focus on input enabled LTSs over $L_I \times L_U$ where Σ, with $\Sigma \notin L_I$, may belong to L_U. Moreover we will let ${}^{\Sigma}\Theta_E$ denote $\Theta_E \cup \{\Sigma\}$.

In Section 5.1 the stuttering semantics is given and its relation with the original semantics of UMLSCs proposed in [15] is addressed in Section 5.2. In Section 5.3, the Conformance Relation is introduced, on which the test case generation algorithm (Section 5.4) is based. Before proceeding with the definition of the stuttering semantics we need some further auxiliary definitions related to the Conformance Relation and to test case generation:

[8] In fact in UML the notion of *deferred events* is introduced in order not to loose events as a consequence of stuttering. In our work we do not take deferred events into consideration.

Definition 34 (*More auxiliary definitions for LTSs*). For LTS $\mathcal{M} = (S, s_{\text{in}}, L, \rightarrow)$, with $L = L_I \times L_U$, $L' = L'_I \times L'_U$, $s \in S$, $Z \subseteq S$, $i \in L'_I$, and $\gamma \in L'^*$:

- The *states* of s *after* γ is the set defined below:

$$(s \;\mathsf{after}\; \gamma) =_{df} \begin{cases} \{s' \mid s \xrightarrow{\gamma} s'\} & \text{if } \gamma \in L^*, \\ \emptyset & \text{otherwise.} \end{cases}$$

- The *output* of Z on i is the set defined below:

$$(\mathsf{out}\; Z\; i) =_{df} \begin{cases} \bigcup_{s \in Z}\{u \in L_U \mid s \xrightarrow{(i,u)}\} & \text{if } i \in L_I, \\ \emptyset & \text{otherwise.} \end{cases}$$

  we let $(\mathsf{OUT}\; s\; \gamma\; i)$ be the set $(\mathsf{out}\,(s \;\mathsf{after}\; \gamma)\; i)$; moreover, we will often denote $(\mathsf{OUT}\; s_{\text{in}}\; \gamma\; i)$ by $(\mathsf{OUT}\; \mathcal{M}\; \gamma\; i)$;
- $\mathcal{M}$ is *input enabled* iff $\forall s \in S, i \in L_I.\ \exists u \in L_U.\ s \xrightarrow{(i,u)}$.

Moreover, for $L = L_I \times L_U$, $L' = L'_I \times L'_U$, $\mathcal{F} \subseteq L^*$, $i \in L'_I$, and $\gamma \in L'^*$

- The traces of $\mathcal{F}$ *after* γ is the set defined below:

$$(\mathcal{F} \;\mathsf{after}^*\; \gamma) =_{df} \begin{cases} \{\gamma' \mid \gamma\gamma' \in \mathcal{F}\} & \text{if } \gamma \in L^*, \\ \emptyset & \text{otherwise.} \end{cases}$$

- The *output* of $\mathcal{F}$ on i is the set defined below:

$$(\mathsf{out}^*\; \mathcal{F}\; i) =_{df} \begin{cases} \{u \in L_U \mid \exists \gamma.\ (i,u)\gamma \in \mathcal{F}\} & \text{if } i \in L_I, \\ \emptyset & \text{otherwise.} \end{cases}$$

  we let $(\mathsf{OUT}^*\; \mathcal{F}\; \gamma\; i)$ be the set $(\mathsf{out}^*\; (\mathcal{F} \;\mathsf{after}^*\; \gamma)\; i)$.

5.1. Stuttering semantics

Definition 35 (*Stuttering semantics*)**.** The stuttering semantics of an HA $H = (F, E, \rho)$ is the LTS over $E \times {}^{\Sigma}\Theta_E$, ${}^{\Sigma}\mathsf{LTS}(H) =_{df} (\text{Conf}_H, \mathcal{C}_{\text{in}}, \rightarrow_\Sigma)$ where (i) Conf_H is the set of configurations, (ii) $\mathcal{C}_{\text{in}} \in \text{Conf}_H$ is the initial configuration, (iii) $\rightarrow_\Sigma \subseteq \text{Conf}_H \times (E \times {}^{\Sigma}\Theta_E) \times \text{Conf}_H$ is the *step*-transition relation defined below, where, as usual, we write $\mathcal{C} \xrightarrow{e/\mathcal{E}}_\Sigma \mathcal{C}'$ for $(\mathcal{C}, (e, \mathcal{E}), \mathcal{C}') \in \rightarrow_\Sigma$.

Definition 36 (*Stuttering semantics transition rules*).

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

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

Fig. 12 shows ${}^{\Sigma}\mathsf{LTS}(H)$, for H as in Fig. 2. For simplicity, several stuttering loops from/to the same state, labeled by $i_1/\Sigma, \ldots, i_k/\Sigma$ have been collapsed to a single loop labeled by $i_1, \ldots, i_k/\Sigma$.

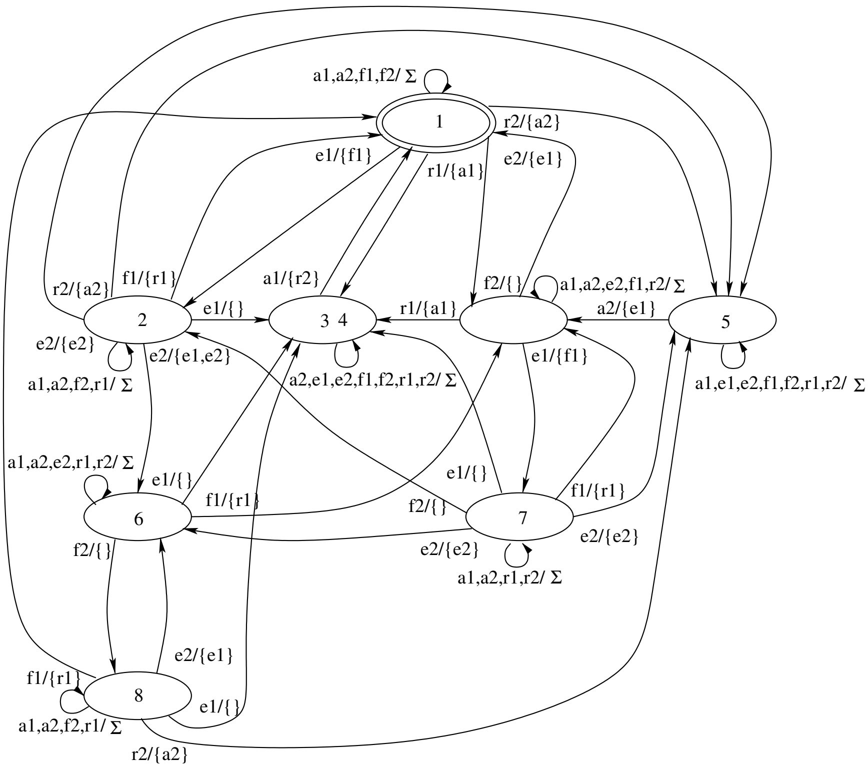


Fig. 12. $^{\Sigma}\mathsf{LTS}(H)$, for H of Fig. 2.

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

Lemma 37. *For HA* $H = (F, E, \rho)$, *all* $\mathcal{C} \in \mathrm{Conf}_H$, *and* $e \in E$ *the following holds*:

(i) $\exists u \in {}^{\Sigma}\Theta_E, \mathcal{C}' \in \mathrm{Conf}_H.\ \mathcal{C} \xrightarrow{e/u}_{\Sigma} \mathcal{C}'$, *i.e.* $^{\Sigma}LTS(H)$ *is* input enabled *over* $E \times {}^{\Sigma}\Theta_E$;

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

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

Finally, it is easy to see that also the stuttering semantics associates a *finite* LTS to each HA.

5.2. Correctness

As in the case of the non-stuttering semantics, also for the stuttering semantics we show the formal link to the original semantics, presented in [15], and recalled in Section 3.2.

We recursively define a specific experimenter, $(\Sigma\mathrm{Queue}(\mathcal{E}_0), {}^{\Sigma}\Theta_E)$, which behaves like a queue and is the same as that used in Section 4.2, except that it has to deal also with Σ. In particular, when receiving Σ from the HA, it disregards it, as it can be seen in Fig. 13.

$$
\begin{aligned}
&\Sigma\mathrm{Queue}(\mathcal{F} : \Theta_E) \triangleq \\
&(\mathsf{Sel}\ \mathcal{F}\ f\ \mathcal{F}') \Rightarrow (f; \lambda X.\ ((X \neq \Sigma \Rightarrow \Sigma\mathrm{Queue}(\mathsf{Join}\ \mathcal{F}'\ X)) \\
&\qquad\qquad\qquad\qquad\qquad + \\
&\qquad\qquad\qquad\qquad\qquad (X = \Sigma \Rightarrow \Sigma\mathrm{Queue}(\mathcal{F}'))))
\end{aligned}
$$

Fig. 13. Definition of $\Sigma\mathrm{Queue}(\mathcal{F})$.

The following proposition establishes the correctness of the stuttering semantics definition. The transition relation in the operational semantics given in [15] is denoted by $\xrightarrow{\mathcal{L}}$.

Proposition 38. *For hierarchical automaton* $H = (F, E, \rho), \mathcal{C}, \mathcal{C}' \in \mathrm{Conf}_H,\ \mathcal{E},\ \mathcal{E}',\ \mathcal{E}'' \in \Theta_E$, *the following holds*: $(\Sigma\,Queue(\mathcal{E})\ ||\ \mathcal{C}) \rightsquigarrow (\Sigma\,Queue(\mathsf{Join}\ \mathcal{E}''\ \mathcal{E}')\ ||\ \mathcal{C}')$ *if and only if* $\exists \mathcal{L}.\ (\mathcal{C}, \mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}', (\mathsf{Join}\ \mathcal{E}''\ \mathcal{E}'))$.

5.3. Conformance relation

In the context of the present work, we assume that a *specification* of system behavior is given in the form of a UMLSC H (in practice we use its HAs representation) and we make reference mainly to the LTS associated to H by the stuttering semantics, i.e. ${}^{\Sigma}\mathsf{LTS}(H)$, over $L = L_I \times L_U$. An *implementation* for H will be modeled by an input-enabled LTS over $L' = L'_I \times L'_U$ (with L'_I not necessarily equal to L_I). Under the above assumptions, for simplicity, we often speak of specifications over L and implementations over L'. We remind the reader that $\Sigma \notin L_I \cup L'_I$ is assumed while $\Sigma \in L_U$ (resp. $\Sigma \in L'_U$) represents stuttering of the specification (resp. implementation). Notice that we do not require that input-enabled LTSs modeling implementations are necessarily generated from UMLSCs. 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 UMLSC. The above assumptions are quite standard in the context of formal conformance theory and its application [41].

In the approach to conformance testing introduced by Tretmans, [42], inputs and outputs are “irregularly” scattered throughout the LTS, and a “quiescence” transition from a state means that in this particular state no output is produced by the system. We remark that, in such an approach, input is not (always) required in order to produce some output. In our setting, there is a clear causal relation between input and related output. They both appear in the same transition. A stuttering transition in a given state—actually a stuttering loop—is labeled by (i, Σ), which means that in that state the system produces no output, or better, does not react at all, *on input* i.

On the basis of the above considerations, with particular reference to the role played by the *input* events of transitions, we give the following definition of our conformance relation. We define it for generic LTSs over i/o-pairs, although we will use it only for input-enabled ones. Finally, we point out that we actually define a *class* of conformance relations, in a similar way as in [41]. The class is indexed by a set $\mathcal{F}$ of traces which determines the discriminatory power of the relation. Such a parametric definition turns out to be of technical help in the definition of the test case generation algorithm in the next section and in the proof of its properties. The definition of the *Conformance Relation* $\sqsubseteq_{\mathsf{co}}^{\mathcal{F}}$ follows:

Definition 39 (*Conformance relations*). For LTSs $\mathcal{M} = (S, s_{\text{in}}, L, \longrightarrow)$, with $L = L_I \times L_U$, $\mathcal{M}' = (S', s'_{\text{in}}, L', \longrightarrow')$, with $L' = L'_I \times L'_U$, and $\mathcal{F} \subseteq (L_I \times L_U)^*$: $\mathcal{M}' \ \sqsubseteq_{\mathsf{co}}^{\mathcal{F}} \ \mathcal{M}$ iff $\forall \gamma \in \mathcal{F}, i \in L_I.\ \mathsf{OUT}\ \mathcal{M}'\ \gamma\ i \subseteq \mathsf{OUT}\ \mathcal{M}\ \gamma\ i$.

In the following we will let $\sqsubseteq_{\mathsf{co}}$ (i.e. "conforms to") denote $\sqsubseteq_{\mathsf{co}}^{(\mathsf{lan}\ \mathcal{M})}$ and we will mainly focus on $\sqsubseteq_{\mathsf{co}}$. Intuitively, $\mathcal{M}' \sqsubseteq_{\mathsf{co}} \mathcal{M}$ means that $\mathcal{M}'$ can never produce an output which could not be produced by $\mathcal{M}$ in the same situation, i.e. after the same i/o sequence *and the same input*. In general, it is not required that $L_I = L'_I$: for partial specifications we have that $L_I \subseteq L'_I$, while for incomplete implementations we have that $L'_I \subseteq L_I$; The case that $L_I \cap L'_I = \emptyset$ does not make so much sense. Notice that when $\Sigma \in L_U$ the above definition implies that $\mathcal{M}'$ may produce no output at all due to stuttering only if $\mathcal{M}$ can do so. This is also the case in [41,42] but its technical definition has been adapted here for UMLSCs. The following lemmas relate the conformance relation with LTS languages.

Lemma 40. *For $\mathcal{M}'$ finite LTS over $L'_I \times L'_U$, $\mathcal{M}$ finite LTS over $L_I \times L_U$, the following holds*: $(\mathsf{lan}\ \mathcal{M}') \subseteq (\mathsf{lan}\ \mathcal{M})$ *implies* $\mathcal{M}' \ \sqsubseteq_{\mathsf{co}} \ \mathcal{M}$.

Lemma 41. *For $\mathcal{M}'$ finite LTS over $L'_I \times L'_U$, $\mathcal{M}$ finite LTS over $L_I \times L_U$, with $L'_I \subseteq L_I$, the following holds*: $\mathcal{M}' \ \sqsubseteq_{\mathsf{co}} \ \mathcal{M}$ *implies* $(\mathsf{lan}\ \mathcal{M}') \subseteq (\mathsf{lan}\ \mathcal{M})$.

The notion of *verdict* is central in conformance testing. A *verdict* is the result of testing a system $\mathcal{M}$ against a test case $\mathcal{T}$, the latter being an experimenter as defined in Section 3. The test is passed if all computations are successful:

Definition 42 (*Verdict*). The *verdict* $\mathcal{V}$ of $\mathcal{T}$ on $\mathcal{M}$ is defined as follows:

$$\mathcal{V}\ \mathcal{T}\ \mathcal{M} =_{df} \begin{cases} \textbf{pass} & \text{if } \perp \notin \text{Result}(\mathcal{T}, \mathcal{M}), \\ \textbf{fail} & \text{otherwise.} \end{cases}$$

A *test suite* is a set of test cases. The *verdict* function is extended to test suites in the obvious way; for test suite $\aleph$

$$\mathcal{V}\ \aleph\ \mathcal{M} =_{df} \begin{cases} \textbf{pass} & \text{if } \forall \mathcal{T} \in \aleph.\ \mathcal{V}\ \mathcal{T}\ \mathcal{M} = \textbf{pass}, \\ \textbf{fail} & \text{otherwise.} \end{cases}$$

The following definition relates test suites to specifications using conformance relations and introduces the notions of sound and exhaustive test suites.

Definition 43 (*Completeness*). Given specification $\mathcal{M}$ and test suite $\aleph$

- $\aleph$ is *sound* w.r.t. $\mathcal{M}$ and $\sqsubseteq_{\mathsf{co}}^{\mathcal{F}}$ iff $\mathcal{M}' \ \sqsubseteq_{\mathsf{co}}^{\mathcal{F}} \ \mathcal{M}$ implies $\mathcal{V}\ \aleph\ \mathcal{M}' = \textbf{pass}$, for all implementations $\mathcal{M}'$;
- $\aleph$ is *exhaustive* w.r.t. $\mathcal{M}$ and $\sqsubseteq_{\mathsf{co}}^{\mathcal{F}}$ iff $\mathcal{V}\ \aleph\ \mathcal{M}' = \textbf{pass}$ implies $\mathcal{M}' \ \sqsubseteq_{\mathsf{co}}^{\mathcal{F}} \mathcal{M}$, for all implementations $\mathcal{M}'$.

We say that a test suite is complete if it is both exhaustive and sound.

5.4. The test case generation algorithm

Once a formal specification of a system has been developed, it is possible to mechanically generate test cases for that specification. The test case generation algorithm TD proposed in this paper is a non-deterministic algorithm which given $L = L_I \times L_U$ and $L' = L'_I \times L'_U$ and $\mathcal{F} \subseteq L^*$, after a *finite* number of recursive calls, returns a test case specification $(\mathcal{U}, L'_U)$ in the test case language introduced in Section 3.3. The definition of the test case generation algorithm TD is given in Fig. 14, where the second component L'_U of experimenter specifications is omitted for notational simplicity. The intuitive behavior of the algorithm is rather simple; at each call, the algorithm generates a single test case. In particular, at each call, it may (non-deterministically) either generate the test which always reports success $(\tau; \mathbf{W}; \delta)$, after which it terminates, or generate a test case as follows. An event e is (non-deterministically) chosen which belongs both to the input alphabet of the specification (L_I) and to that of the implementation (L'_I) and such that the set $\mathsf{out}^*\ \mathcal{F}\ e = \{u_1, \ldots, u_k\}$ is non-empty (notice that such an e exists when dealing with input enabled LTS over i/o-pairs associated to UMLSCs). Intuitively, $u_1, \ldots, u_k$ are the expected correct values for the output of the implementation under test as reaction to input e. Consequently, a test case is generated which first sends e to the implementation and then, if the output of the implementation does not match any of the expected values $u_1, \ldots, u_k$, it stops without reporting success, otherwise, assuming that the output of the implementation is u_j, it continues as $\mathcal{U}_j$. Notice that test case $\mathcal{U}_j$ is generated by a recursive call of the algorithm. Different test cases can be generated from $\mathcal{F}$, L and L' by repeating the procedure defined in Fig. 14. This way, the set of all test cases from $\mathcal{F}$, L and L' can be generated. Notice that, by construction, test cases generated by TD have a tree-like structure; there is no looping possibility in their execution. The following proposition establishes finiteness of test cases generated by the algorithm on the stuttering semantics of HAs.

Proposition 44. *For every HA H with $^{\Sigma}\mathsf{LTS}(H)$ over i/o-pair set L, and i/o-pair set L', every test case $\mathcal{U} \in \mathsf{TD}_{L,L'}\ (\mathsf{lan}(^{\Sigma}\mathsf{LTS}(H)))$ is finite.*

Typically $\mathsf{lan}(^{\Sigma}\mathsf{LTS}(H))$ is an infinite set. This does not affect the effectiveness of TD since, at each recursive step, it uses *only* the first elements of the traces in the set, postponing the use of their tails to the next recursive calls. Thus, proper lazy techniques can be used for the evaluation of $\mathsf{lan}(^{\Sigma}\mathsf{LTS}(H))$. Notice also that the set of all test cases generated using $\mathsf{TD}_{L,L'}$ on $\mathsf{lan}(^{\Sigma}\mathsf{LTS}(H))$ is infinite. Each individual test case is however finite. As an immediate consequence of the above lemma and the fact that the test cases generated by the algorithm do not contain loops, we have that all computations involving test cases in $\mathsf{TD}_{L,L'}\ (\mathsf{lan}(^{\Sigma}\mathsf{LTS}(H)))$ are finite.

The following theorem establishes completeness of the test case generation algorithm, when applied to (the language of) a specification $^{\Sigma}\mathsf{LTS}(H)$:

Theorem 45. *For every HA H with $^{\Sigma}\mathsf{LTS}(H)$ over $L = L_I \times L_U$, and set $L' = L'_I \times L'_U$, the test suite $\mathsf{TD}_{L,L'}\ (\mathsf{lan}(^{\Sigma}\mathsf{LTS}(H)))$ is* complete *w.r.t. $^{\Sigma}\mathsf{LTS}(H)$ and $\sqsubseteq_{\mathsf{co}}$.*

The above important result means that if a test case generated by the algorithm for a certain specification H reports a failure when running against an implementation, then we can be sure that the latter does not conform to the specification H; moreover, if an implementation does not conform to specification H, then a test case can be generated by the algorithm which will report failure when executed against such an implementation.

For $L = L_I \times L_U$ and $L' = L'_I \times L'_U$ we define the following non-deterministic algorithm which, given set $\mathcal{F} \subseteq L^*$, after a *finite* number of recursive calls, returns a test case in the test language.

$\mathsf{TD}_{L,L'}\ \mathcal{F}\ :=$

Non-deterministically choose between options (1) and (2) below

1) generate "$\tau; \mathcal{W}$"

2) generate "$e; \lambda x.\ (x = u_1 \ \Rightarrow\ \mathcal{U}_1$

$$\begin{array}{l} + \\ \vdots \\ + \\ x = u_k \ \Rightarrow\ \mathcal{U}_k \\ + \\ x \notin \{u_1, \ldots, u_k\} \ \Rightarrow\ \delta\text{''} \\ ) \end{array}$$

where:

$\mathcal{W}$ is the experimenter $\mathbf{W}; \delta$ which always experiences *success*;

e is non-deterministically chosen in $L_I \cap L'_I$

such that $\mathsf{out}^*\ \mathcal{F}\ e = \{u_1, \ldots, u_k\} \neq \emptyset$, and

$\mathcal{U}_j \in \mathsf{TD}_{L,L'}\ (\mathcal{F}\ \mathsf{after}^*\ (e, u_j))$ for $j = 1, \ldots, k$

Fig. 14. The test case generation algorithm.

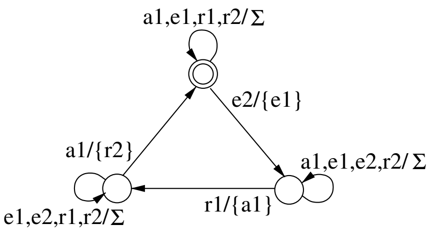


Fig. 15. An implementation of the HA of Fig. 2.

$$\begin{array}{l} \mathcal{U}_1 \triangleq e_2; \lambda x.((x = \{e_1\} \Rightarrow \mathcal{U}_2) + (x \notin \{\{e_1\}\} \Rightarrow \delta)) \\ \mathcal{U}_2 \triangleq r_1; \lambda x.\ (x = \{a_1\} \Rightarrow \mathcal{U}_3 \\ \qquad\qquad + \\ \qquad\qquad x = \Sigma \Rightarrow \mathcal{U}_4 \\ \qquad\qquad + \\ \qquad\qquad x \notin \{\{a_1\}, \Sigma\} \Rightarrow \delta \\ \qquad\qquad) \\ \mathcal{U}_3 \triangleq a_1; \lambda x.((x = \{r_2\} \Rightarrow \mathcal{U}_4) + (x \notin \{\{r_2\}\} \Rightarrow \delta)) \\ \mathcal{U}_4 \triangleq \tau; \mathbf{W}; \delta \end{array}$$

Fig. 16. A test case generated from the running example.

$$\mathcal{U}_5 \triangleq r_1; \lambda x.((x = \{a_1\} \Rightarrow \mathcal{U}_4) + (x \notin \{\{a_1\}\} \Rightarrow \delta))$$

Fig. 17. Another test case generated from the running example.

We close this section with an application of the test case derivation algorithm to our running example. Let us consider again the specification $\mathcal{M}$ of Fig. 2 and the (obviously incomplete) implementation $\mathcal{M}'$ over $L'_I \times L'_U$ with $L'_I = \{a_1, e_1, e_2, r_1, r_2\}$ and $L'_U = \{\Sigma, \{a_1\}, \{e_1\}, \{r_2\}\}$ given in Fig. 15.

We can apply the algorithm in order to obtain, among others, the test case $\mathcal{U}_1$ shown in Fig. 16. It is easy to see that $\mathcal{V}\,\mathcal{U}_1\,\mathcal{M}' = \mathbf{pass}$. On the other hand, $\mathcal{M}' \not\sqsubseteq_{\mathsf{co}} \mathcal{M}$, and this can be checked using the test case $\mathcal{U}_5$ shown in Fig. 17, which is also derived using the algorithm. Clearly $\mathcal{V}\,\mathcal{U}_5\mathcal{M}' = \mathbf{fail}$.

6. Relating testing and conformance relations

In this section we report the major results concerning the relationship between the stuttering and the non-stuttering semantics and the relationship between the Testing Preorders (and Equivalence) and the Conformance Relation. We shall make explicit reference to HAs representing UMLSCs. In particular, in the following we shall assume that for each (HA representing a specific) UMLSC H the set of events E on which H is defined, i.e. its *alphabet*, is given explicitly. Set E will include all the events occurring in H. Under the above conditions, we will speak of UMLSC H *on* E. Moreover, in the case of specifications where the behavior of the system is only partially specified, there might be elements of E which do not occur in H.

It is worth reminding the reader here that both $\mathsf{LTS}(H)$ and ${}^{\Sigma}\mathsf{LTS}(H)$ have a finite number of states and a finite number of step-transitions. Furthermore, they 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 ${}^{\Sigma}\mathsf{LTS}(H)$, thus avoiding confusion about which LTS we are dealing with.

6.1. Relating the stuttering and the non-stuttering semantics

In this section we take a closer look at the formal relationship between the stuttering semantics and the non-stuttering one.

Theorem 46. *For all HAs $H = (F, E, \rho)$, $e \in E$ and $\mathcal{C} \in \mathrm{Conf}_H$ the following holds*:

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

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

Thus the two semantics generate the same step-transitions, except for stuttering, i.e. when the machine does not accept the current event e in the current state. This refusal is modeled (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, whose step-relation is recalled in Definition 3, simply generates step-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 HA always includes the explicit specification of the input set E, specially when we compare different HAs on the basis of testing/conformance relations, as in the following sections. The following is a useful corollary to Theorem 46:

Corollary 47. *For all HAs* $H = (F, E, \rho)$, $\mathcal{C}, \mathcal{C}' \in \mathrm{Conf}_H, \gamma \in (E \times \Theta_E)^* : \mathcal{C} \xrightarrow{\gamma} \mathcal{C}'$ *iff* ${}^{\Sigma}\mathcal{C} \xrightarrow{\gamma}_{\Sigma} {}^{\Sigma}\mathcal{C}'$.

We close this section with a lemma relating the languages of $\mathsf{LTS}(H)$ and ${}^{\Sigma}\mathsf{LTS}(H)$, where we use the following operator (_ _), where $\gamma \setminus \Sigma$ is equal to γ where all occurrences of Σ are removed.

Definition 48 ($\gamma \setminus \Sigma$). For $\gamma \in (E \times {}^{\Sigma}\Theta_E)$ we define $\gamma \setminus \Sigma$ as follows:

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

Lemma 49. *For HA* $H = (F, E, \rho)$, *all* $\mathcal{C}, \mathcal{C}' \in \mathrm{Conf}_H$, $\gamma \in (E \times {}^{\Sigma}\Theta_E)$, *the following holds*:
(i) ${}^{\Sigma}\mathcal{C} \xrightarrow{\gamma}_{\Sigma} {}^{\Sigma}\mathcal{C}'$ *implies* $\mathcal{C} \xrightarrow{\gamma \setminus \Sigma} \mathcal{C}'$;
(ii) $\gamma \in \mathit{lan}\ {}^{\Sigma}\mathit{LTS}(H)$ *implies* $\gamma \setminus \Sigma \in (\mathit{lan}\ \mathit{LTS}(H))$;
(iii) $(\mathit{lan}\ \mathit{LTS}(H)) \subseteq (\mathit{lan}\ {}^{\Sigma}\mathit{LTS}(H))$.

6.2. Testing preorders and the conformance relation

The following two lemmas show that, under proper conditions, the Conformance Relation is *stronger* than the *MAY* and *MUST* preorders.

Lemma 50. *For all HAs H on E and* H' *on* $E' \subseteq E$ *the following holds*: ${}^{\Sigma}\mathit{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathit{LTS}(H)$ *implies* $\mathit{LTS}(H') \sqsubseteq_{MAY} \mathit{LTS}(H)$.

Lemma 51. *For all HAs H and* H' *on E the following holds*:
${}^{\Sigma}\mathit{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathit{LTS}(H)$ *implies* $\mathit{LTS}(H) \sqsubseteq_{MUST} \mathit{LTS}(H')$.

Notice that Lemma 50 holds also for incomplete implementations ($E' \subset E$), but it requires the specification not to be partial w.r.t. the implementation, which would imply $E \subset E'$. The condition $E' \subseteq E$ is indeed essential, as shown by the following example.

Example 52. Let $E = \{a\} \subseteq \{a, b\} = E'$, with H (resp. H') as in Fig. 18(a) (resp. (a′)), $\mathsf{LTS}(H)$ (resp. $\mathsf{LTS}(H')$) as in Fig. 18(b) (resp. (b′)) and ${}^{\Sigma}\mathsf{LTS}(H)$ (resp. ${}^{\Sigma}\mathsf{LTS}(H')$) as

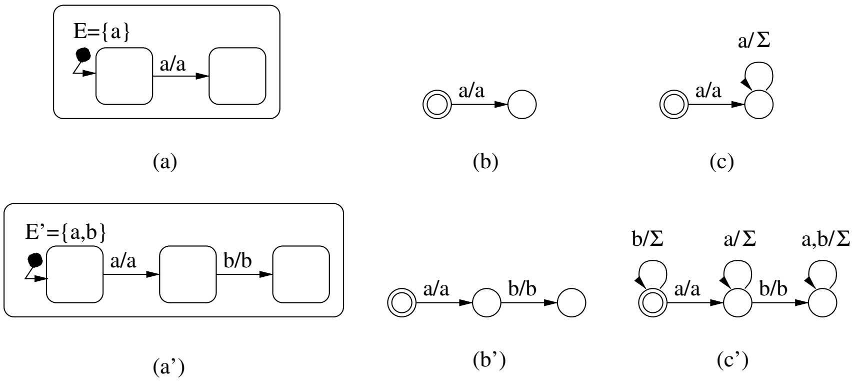


Fig. 18. Example 52.

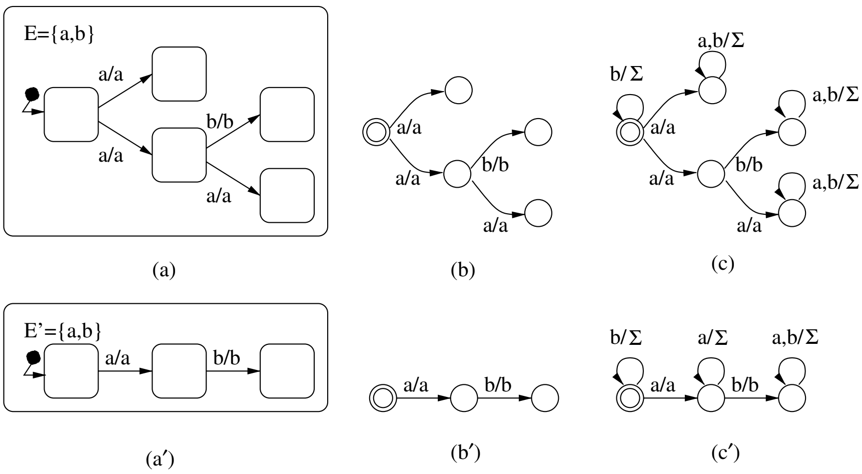


Fig. 19. Example 53.

in Fig. 18(c) (resp. (c′)). Clearly ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$, but $\mathsf{LTS}(H') \underset{\sim}{\sqsubset}_{MAY} \mathsf{LTS}(H)$ does not hold. In fact, from Theorem 21(a), we know that $(a, a)(b, b) \in \mathsf{lan}\ \mathsf{LTS}(H') \setminus \mathsf{lan}\ \mathsf{LTS}(H)$.

Notice furthermore that in Lemma 50 the implication is strictly one way as shown by the following example.

Example 53. Let $E = E' = \{a, b\}$ with H (resp. H') as in Fig. 19(a) (resp. (a′)), $\mathsf{LTS}(H)$ (resp. $\mathsf{LTS}(H')$) as in Fig. 19(b) (resp. (b′)) and ${}^{\Sigma}\mathsf{LTS}(H)$ (resp. ${}^{\Sigma}\mathsf{LTS}(H')$) as in Fig. 19(c) (resp. (c′)). We have $\mathsf{LTS}(H') \underset{\sim}{\sqsubset}_{MAY} \mathsf{LTS}(H)$ but ${}^{\Sigma}\mathsf{LTS}(H') \not\sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$ since $\mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H')\ (a, a)(a, \Sigma)\ b = \{b\} \not\subseteq \{\Sigma\} = \mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H)\ (a, a)(a, \Sigma)\ b$.

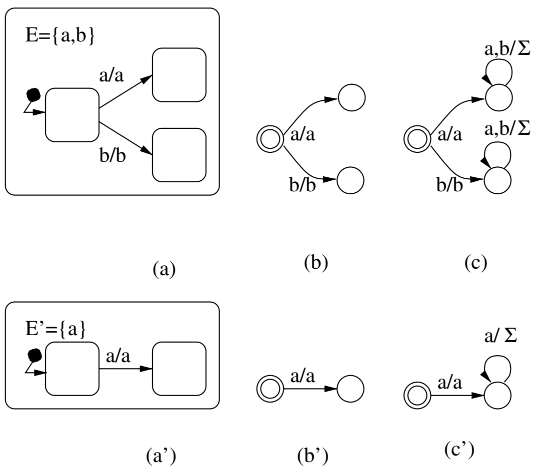


Fig. 20. Example 55.

Notice that in Lemma 51 we require that both H and H' have the same input set E. The following examples show that for H on E and H' on E' neither $E \subseteq E'$ alone nor $E' \subseteq E$ alone is enough:

Example 54. Let H and H' as in Example 52. It is easy to see that it is not the case that $\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MUST} \mathsf{LTS}(H')$. In fact $(a, a)(b, b) \in \mathsf{lan}\ \mathsf{LTS}(H') \setminus \mathsf{lan}\ \mathsf{LTS}(H)$ (see Corollary 83).

Example 55. Let $E = \{a, b\} \supseteq \{a\} = E'$ with H (resp. H') as in Fig. 20(a) (resp. (a')), $\mathsf{LTS}(H)$ (resp. $\mathsf{LTS}(H')$) as in Fig. 20(b) (resp. (b')) and ${}^{\Sigma}\mathsf{LTS}(H)$ (resp. ${}^{\Sigma}\mathsf{LTS}(H')$) as in Fig. 20(c) (resp. (c')). Clearly ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$, but $\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MUST} \mathsf{LTS}(H')$ does not hold since $\mathsf{AS}\ \mathsf{LTS}(H')\ \epsilon = \{\{(a, a)\}\}$ and $\mathsf{AS}\ \mathsf{LTS}(H)\ \epsilon = \{\{(a, a), (b, b)\}\}$ which implies $mfs(\ \mathbf{c}\ (\mathsf{AS}\ \mathsf{LTS}(H')\ \epsilon)) \not\subset\!\!\subset mfs(\ \mathbf{c}\ (\mathsf{AS}\ \mathsf{LTS}(H)\ \epsilon))$.

Notice furthermore that in Lemma 51 the implication is strictly one way as shown by the following:

Example 56. Let H and H' as in Example 53. $\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MUST} \mathsf{LTS}(H')$, but we have seen that ${}^{\Sigma}\mathsf{LTS}(H') \not\sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$.

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

Example 57. Let $E = E' = \{a, b\}$ with H (resp. H') as in Fig. 21(a) (resp. (a')), $\mathsf{LTS}(H)$ (resp. $\mathsf{LTS}(H')$) as in Fig. 21(b) (resp. (b')) and ${}^{\Sigma}\mathsf{LTS}(H)$ (resp. ${}^{\Sigma}\mathsf{LTS}(H')$) as in Fig. 21(c) (resp. (c')). We have $\mathsf{LTS}(H) \underset{\sim}{\sqsubset} \mathsf{LTS}(H')$ since $\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MAY} \mathsf{LTS}(H')$ (actually $\mathsf{lan}\ \mathsf{LTS}(H) = \mathsf{lan}\ \mathsf{LTS}(H')$) and $\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MUST} \mathsf{LTS}(H')$ but ${}^{\Sigma}\mathsf{LTS}(H') \not\sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$ since the following holds: $\mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H')\ (a, a)(a, \Sigma)\ b = \{b\} \not\subseteq \{\Sigma\} = \mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H)\ (a, a)(a, \Sigma)\ b$.

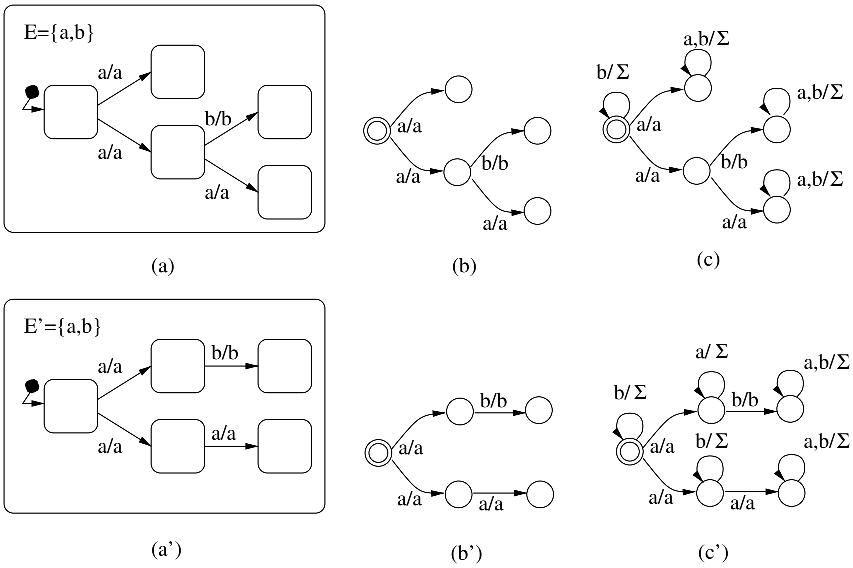


Fig. 21. Example 57.

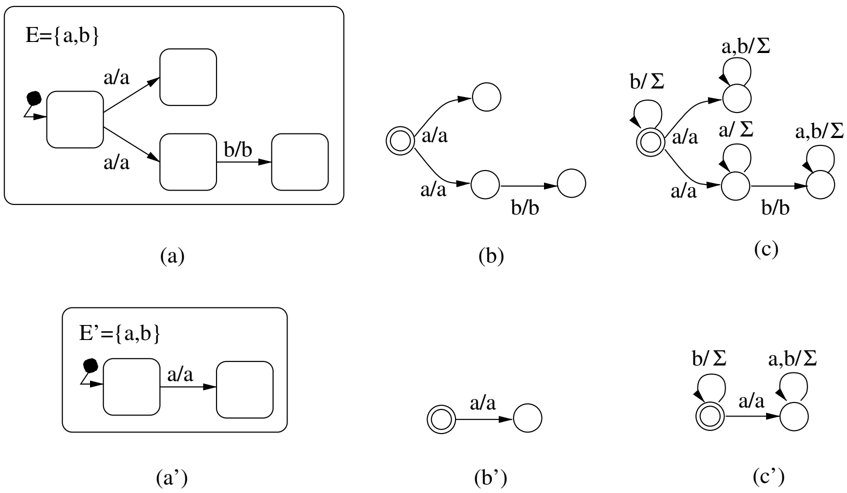


Fig. 22. Example 58.

Example 58. Let $E = E' = \{a, b\}$ with H (resp. H') as in Fig. 22(a) (resp. (a′)), $\mathsf{LTS}(H)$ (resp. $\mathsf{LTS}(H')$) as in Fig. 22(b) (resp. (b′)) and ${}^{\Sigma}\mathsf{LTS}(H)$ (resp. ${}^{\Sigma}\mathsf{LTS}(H')$) as in Fig. 22(c) (resp. (c′)). We have ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$ but $\mathsf{LTS}(H) \underset{\sim}{\sqsubset} \mathsf{LTS}(H')$ does not hold since $\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MAY} \mathsf{LTS}(H')$ does not hold: $(a, a)(b, b) \in \mathsf{lan}\ \mathsf{LTS}(H) \setminus \mathsf{lan}\ \mathsf{LTS}(H')$.

Finally, the following two examples show that testing equivalence over the non-stuttering semantics is not strong enough for detecting LTS's capability of refusing to react and,

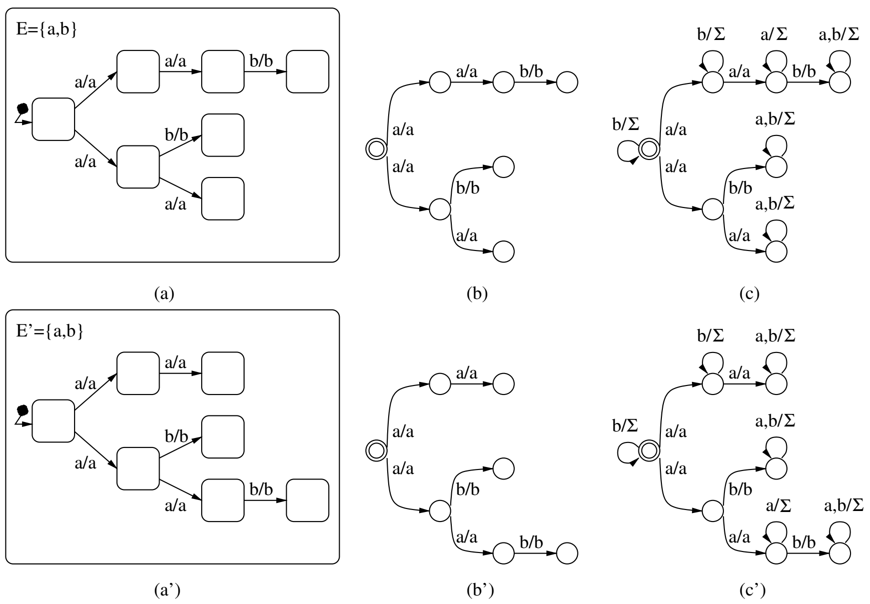


Fig. 23. Example 59.

consequently, for discriminating among them on such basis (Example 59). This in turn implies that testing equivalence does not enjoy substitutivity properties with respect to $\sqsubseteq_{\mathsf{co}}$ (Example 60).

Example 59. Let $E = E' = \{a, b\}$ with H (resp. H') as in Fig. 23(a) (resp. (a′)), $\mathsf{LTS}(H)$ (resp. $\mathsf{LTS}(H')$) as in Fig. 23(b) (resp. (b′)) and $^{\Sigma}\mathsf{LTS}(H)$ (resp. $^{\Sigma}\mathsf{LTS}(H')$) as in Fig. 23(c) (resp. (c′)). We have $\mathsf{LTS}(H) \eqsim \mathsf{LTS}(H')$ but $(a, a)(b, \Sigma)(a, a)(b, b)$ is an element of $\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H) \setminus \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H')$ and $(a, a)(b, \Sigma)(a, a)(b, \Sigma)$ is an element of $\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \setminus \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$.

Example 60. Take H and H' as in Example 59 and let $H'' = H$. From Example 59 we know that $\mathsf{LTS}(H'') \eqsim \mathsf{LTS}(H')$ and trivially $^{\Sigma}\mathsf{LTS}(H'') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$. On the other hand, $^{\Sigma}\mathsf{LTS}(H') \not\sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$; in fact we have that $\mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H')\ (a, a)(b, \Sigma)(a, a)\ b = \{\Sigma\} \not\subseteq \mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H)\ (a, a)(b, \Sigma)(a, a)\ b$ because the latter is equal to $\{b\}$. Similarly, we have that clearly $^{\Sigma}\mathsf{LTS}(H) \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H'')$; but $^{\Sigma}\mathsf{LTS}(H) \not\sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H')$ since $\mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H)\ (a, a)(b, \Sigma)(a, a)\ b$ is the set $\{b\}$ which is not a subset of $\mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H')\ (a, a)(b, \Sigma)(a, a)\ b$, since the latter is the set $\{\Sigma\}$.

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 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 to detect absence of reaction and to take proper actions when this

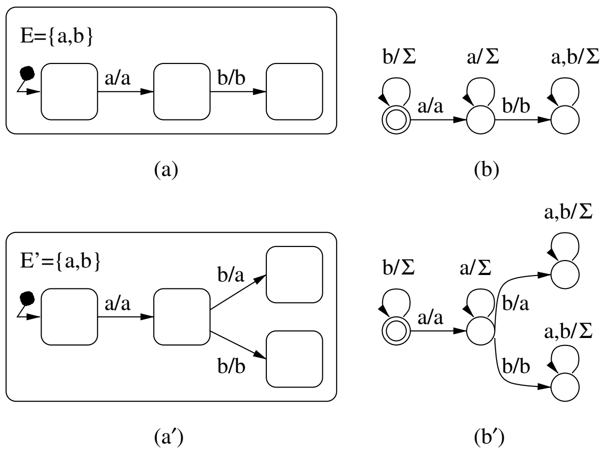


Fig. 24. Example 64.

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 61 and 62 below:

Lemma 61. *For all HAs H on E and H' on E' the following holds*:
(i) ${}^{\Sigma}LTS(H) \sqsubseteq_{MAY} {}^{\Sigma}LTS(H')$ *implies* ${}^{\Sigma}LTS(H) \sqsubseteq_{co} {}^{\Sigma}LTS(H')$;
(ii) ${}^{\Sigma}LTS(H) \sqsubseteq_{co} {}^{\Sigma}LTS(H')$ *and* $E \subseteq E'$ *implies* ${}^{\Sigma}LTS(H) \sqsubseteq_{MAY} {}^{\Sigma}LTS(H')$.

Lemma 62. *For all HAs H on E and H' on E' the following holds*:
${}^{\Sigma}LTS(H) \sqsubseteq_{MUST} {}^{\Sigma}LTS(H')$ *implies* ${}^{\Sigma}LTS(H') \sqsubseteq_{co} {}^{\Sigma}LTS(H)$.

Notice that in Lemma 62 the implication is strictly one way as shown by the following:

Example 63. Let H and H' as in Example 52. We know from that example that ${}^{\Sigma}\mathrm{LTS}(H') \sqsubseteq_{co} {}^{\Sigma}\mathrm{LTS}(H)$. On the other hand it is easy to see that ${}^{\Sigma}\mathrm{LTS}(H) \sqsubseteq_{MUST} {}^{\Sigma}\mathrm{LTS}(H')$ does not hold, since $(\mathsf{lan}\ {}^{\Sigma}\mathrm{LTS}(H')) \not\subseteq (\mathsf{lan}\ {}^{\Sigma}\mathrm{LTS}(H))$ and this would violate Corollary 83.

Finally notice that in general ${}^{\Sigma}\mathrm{LTS}(H) \sqsubseteq_{co} {}^{\Sigma}\mathrm{LTS}(H')$ does *not* imply that ${}^{\Sigma}\mathrm{LTS}(H) \sqsubseteq_{MUST} {}^{\Sigma}\mathrm{LTS}(H')$, as shown by the following:

Example 64. Let $E = E' = \{a, b\}$, with H (resp. H') as in Fig. 24(a) (resp. (a′)) and ${}^{\Sigma}\mathrm{LTS}(H)$ (resp. ${}^{\Sigma}\mathrm{LTS}(H')$) as in Fig. 24(b) (resp. (b′)). Clearly ${}^{\Sigma}\mathrm{LTS}(H) \sqsubseteq_{co} {}^{\Sigma}\mathrm{LTS}(H')$, but it is easy to see that ${}^{\Sigma}\mathrm{LTS}(H) \sqsubseteq_{MUST} {}^{\Sigma}\mathrm{LTS}(H')$ does not hold, since $(\mathsf{lan}\ {}^{\Sigma}\mathrm{LTS}(H')) \not\subseteq (\mathsf{lan}\ {}^{\Sigma}\mathrm{LTS}(H))$ and this would violate Corollary 83.

This last remark shows that in the stuttering semantics, the testing preorder is strictly stronger than the conformance relation.

The following theorem establishes the adequacy of the testing relations based on the stuttering semantics for the conformance relation. On an intuitive level, it is worth pointing out that point (iii) of the theorem essentially states that if an implementation is "less non-deterministic" than one conforming to a specification, then it also conforms to the specification. Similarly, point (vi) says that if a specification is "more non-deterministic" than one to which an implementation conforms, than the implementation will also conform to this specification.

Theorem 65. *For all HAs H on E, H' on E' and H'' on E'', with $E' \subseteq E$, the following holds*:

(i) ${}^{\Sigma}\mathsf{LTS}(H'') \underset{\sim}{\sqsubset}_{MAY} {}^{\Sigma}\mathsf{LTS}(H') \wedge {}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$ *implies* ${}^{\Sigma}\mathsf{LTS}(H'') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$;

(ii) ${}^{\Sigma}\mathsf{LTS}(H') \underset{\sim}{\sqsubset}_{MUST} {}^{\Sigma}\mathsf{LTS}(H'') \wedge {}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$ *implies* ${}^{\Sigma}\mathsf{LTS}(H'') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$;

(iii) ${}^{\Sigma}\mathsf{LTS}(H') \underset{\sim}{\sqsubset} {}^{\Sigma}\mathsf{LTS}(H'') \wedge {}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$ *implies* ${}^{\Sigma}\mathsf{LTS}(H'') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$;

(iv) ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H) \wedge {}^{\Sigma}\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MAY} {}^{\Sigma}\mathsf{LTS}(H'')$ *implies* ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H'')$;

(v) ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H) \wedge {}^{\Sigma}\mathsf{LTS}(H'') \underset{\sim}{\sqsubset}_{MUST} {}^{\Sigma}\mathsf{LTS}(H)$ *implies* ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H'')$;

(vi) ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H) \wedge {}^{\Sigma}\mathsf{LTS}(H'') \underset{\sim}{\sqsubset} {}^{\Sigma}\mathsf{LTS}(H)$ *implies* ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H'')$.

A useful corollary of the above theorem states the substitutivity properties of $\eqsim$ with respect to $\sqsubseteq_{\mathsf{co}}$.

Corollary 66. *For all HAs H on E, H' on E' and H'' on E'', with $E' \subseteq E$ the following holds*:

(i) ${}^{\Sigma}\mathsf{LTS}(H') \eqsim {}^{\Sigma}\mathsf{LTS}(H'') \wedge {}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$ *implies* ${}^{\Sigma}\mathsf{LTS}(H'') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$;

(ii) ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H) \wedge {}^{\Sigma}\mathsf{LTS}(H'') \eqsim {}^{\Sigma}\mathsf{LTS}(H)$ *implies* ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H'')$.

We close this section with the following three propositions relating the non-stuttering semantics and the stuttering one via the testing relations in the way one would expect:

Proposition 67. *For all HAs H on E and H' on E' the following holds*:
${}^{\Sigma}\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MAY} {}^{\Sigma}\mathsf{LTS}(H')$ *implies* $\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MAY} \mathsf{LTS}(H')$.

Proposition 68. *For all HAs H, H' on E the following holds*:
${}^{\Sigma}\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MUST} {}^{\Sigma}\mathsf{LTS}(H')$ *implies* $\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MUST} \mathsf{LTS}(H')$.

Proposition 69. *For all HAs H, H' on E the following holds*:
${}^{\Sigma}\mathsf{LTS}(H) \underset{\sim}{\sqsubset} {}^{\Sigma}\mathsf{LTS}(H')$ *implies* $\mathsf{LTS}(H) \underset{\sim}{\sqsubset} \mathsf{LTS}(H')$.

Notice again that the above implications are strictly one way, as can be seen from Example 59, using Theorem 21(a) and Corollary 83.

7. Conclusions

The main contribution of the present paper is a theoretical framework for testing theory and verification as well as test case generation in a conformance testing setting. We presented a testing theory for UML Statecharts (UMLSCs) with an algorithm for automatic verification of testing equivalence—based on a formal "non-stuttering" semantics—and a conformance relation for UMLSCs as well as an algorithm for test case generation—based on a formal "stuttering" semantics. The automatic verification algorithm has been proved correct and the test case generation algorithm was proved complete. Both proofs are presented in this paper. The formal relationships between the stuttering and non-stuttering semantics were investigated and all related proofs provided. In particular, we proved that the non-stuttering semantics for the testing preorders is not a good choice when also conformance is an issue. In fact we showed that the conformance relation for UMLSCs is strictly stronger than the reverse of the *MUST* preorder based on the non-stuttering semantics, and then also stronger than the associated *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. 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, i.e. stuttering and behaving accordingly. We showed that, in this case, the *MAY* (resp. *MUST*) preorder is stronger than $\sqsubseteq_{\mathtt{co}}$ (resp. inverse of $\sqsubseteq_{\mathtt{co}}$). As a consequence, one can replace testing equivalent specifications and implementations still preserving their conformance relation. More specifically, if an implementation is "less non-deterministic" than one conforming to a specification, then it also conforms to the specification. Similarly, if a specification is "more non-deterministic" than one to which an implementation conforms, then the implementation will also conform to this specification. This is an important result in the framework of a system development approach in which e.g. implementations are replaced with equivalent or "less non-deterministic" ones in a stepwise manner, still maintaining the conformance relation with their specifications.

Our work represents also a contribution to the investigation on the relationship between notions developed in the area of state-based, object-oriented, programming, like subtyping/sub-classes, and behavioural relations. In [6] it is argued that behavioral relations, and in particular testing preorders, may form the basis for studying the above-mentioned notions. Our results open the way to the extension to UML of the approach presented in [6].

With respect to the testing equivalence verification algorithm, the determinization phase of the testing equivalence verification algorithm may take exponential time, but this should not surprise the reader because it has been proved that the verification of testing equivalence is a PSPACE-complete problem [4]. Other equivalences are easier to verify but they may be too strong, like e.g. bisimulation equivalence itself which distinguishes machines also on the basis of their internal structure and not only on the basis of their interaction with the external environment. Anyway, the fact that the semantics generate finite LTSs over i/o-pairs allows us to perform bisimulation equivalence verification directly on such LTSs, should this turn up useful.

In order to use the test generation algorithm in practice proper *test selection* strategies are needed which will be subject of future work. Some work on test selection in a formal test derivation framework is already present in the literature (see, e.g. [7,2,12]), and in particular random test case selection seems to be a promising option. In fact it nicely fits with the structure of our algorithm; what is needed is to replace non-deterministic choices with random, coin-flipping, ones. Moreover, random test selection is receiving more and more attention due to the high coverage that it can provide, using efficient automated tools. Another promising line of research is the use of model-checking techniques for enhancing automatic test case generation, which we are currently investigating [10]. Closely related to the above research lines is the area of efficient implementation of test generation and selection algorithms. There are already tools available to that purpose, e.g. AutoFocus [37] and TGV/AGEDIS [39], and one of the next steps will be an investigation on the possibility of providing a connection between our work and such tools.

In the present paper we made no assumption on how test cases are "implemented", i.e. on their actual presentation. They might be represented again as UMLSCs or as UML Sequence Diagrams or just as code in a proper programming language. This last possibility could allow for the implementation of test runs using proper automatic tools, to be integrated with the test case generation tools, which is our ultimate goal.

Another line of future research deals with the extension of the results presented in the present paper to UML specifications consisting of *collections* of UMLSCs interacting via queues [15], which brings to *distributed* testing. The use of a test language like the one proposed in the present paper, which is easy to extend in order to allow control communication between the experimenters to take place, greatly facilitates the task of specifying complex distributed test cases and developing a suitable extension of testing theory to the distributed case.

A further useful extension is the introduction of data values and variables in UMLSCs. We have already a semantics definition for such an extension, fully developed in the context of the the PRIDE project [20]. Of course (infinite) data sets pose further problems in the test selection procedures.

The results addressed in the present paper have been originally proposed in [25,26,16], although in isolation, while in the present paper they have been dealt with in a uniform framework and notation. Moreover all proofs, which were omitted in the above-mentioned papers, are provided in the present paper.

Acknowledgments

The authors wish to thank the anonymous reviewers for the detailed and precious comments they provided in order to improve the paper.

Appendix A. Hierarchical automata

The first step of our approach is a purely syntactical one and consists in translating UMLSCs into what are usually called Hierarchical Automata (HAs). HAs can be seen as an *abstract syntax* for UMLSCs in the sense that they abstract from the purely syntactical/graphical details and describe only the essential aspects of the statechart. They are composed of simple sequential automata related by a *refinement function*. A state is mapped via the refinement function into the set of (parallel) automata which refine it. The translation from UMLSCs to HAs has been dealt with in [24]. In this appendix we recall the notion of HAs as defined in [32,24].

A.1. Basic definitions and semantics

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

Definition 70 (*Sequential Automata*). A sequential automaton A is a 4-tuple $(\sigma_A, s_A^0, \lambda_A, \delta_A)$ where σ_A is a finite set of *states* with $s_A^0 \in \sigma_A$ the *initial state*, λ_A is a finite set of *transition labels*, with $\lambda_A \cap \sigma_A = \emptyset$ and $\delta_A \subseteq \sigma_A \times \lambda_A \times \sigma_A$ is the *transition relation*.

In the context of HAs, the labels in λ_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 $t = (s, (sr, ev, g, ac, td), s')$, $SRC\ t = s$, $TGT\ t = s'$, $SR\ t = sr$, $EV\ t = ev$, $G\ t = g$, $AC\ t = ac$, $TD\ t = td$. Their meaning is described in [24]. Hierarchical Automata are defined as follows:

Definition 71 (*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 $A_{root} \notin \bigcup_{X \in rng\ \rho} X$, (ii) every non-root automaton has exactly one ancestor state: $\bigcup_{X \in rng\ \rho} X = F \setminus \{A_{root}\}$ 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: $\forall S \subseteq \bigcup_{A \in F} \sigma_A.\ \exists s \in S.\ S \cap \bigcup_{A \in \rho s} \sigma_A = \emptyset$.

We say that a state s for which $\rho\ s = \emptyset$ holds is a *basic* state.

The notion of *conflict* between transitions needs to be extended in order to deal with state hierarchy. When transitions t and t' are in conflict we write $t \# t'$. The complete formal definition of conflict for HAs can be found in [24,15] where also the notion of *priority* for (conflicting) transitions is defined. Intuitively transitions coming from deeper states have higher priority. For the purposes of the present paper it is sufficient to say that priorities form a partial order. We let $\pi\ t$ denote the priority of transition t and $\pi\ t \sqsubseteq \pi\ t'$ mean that t has lower priority than (the same priority as) t'. In the sequel we will be concerned only with HAs.

In the sequel we implicitly make reference to a generic HA $H = (F, E, \rho)$. Moreover, we also assume implicitly that each transition of each sequential automaton in F is uniquely identified by its label. This can always be obtained by adding unique identifiers to labels whenever necessary. Every sequential automaton $A \in F$ characterizes 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.

The following definition characterizes a couple of useful functions:

Definition 72. For $A \in F$ the *automata*, *states*, and *transitions under* A are defined respectively as

(i) $\mathsf{Au}\ A =_{df} \{A\} \cup \left(\bigcup_{A' \in \left(\bigcup_{s \in \sigma_A} (\rho_A s)\right)} (\mathsf{Au}\ A')\right)$;

(ii) $\mathsf{St}\ A =_{df} \bigcup_{A' \in \mathsf{Au}\ A} \sigma_{A'}$;

(iii) $\mathsf{Tr}\ A =_{df} \bigcup_{A' \in \mathsf{Au}\ A} \delta_{A'}$.

The definition of sub-HA follows:

Definition 73 (*Sub Hierarchical Automata*). For $A \in F$, the HA characterized by A is the triple (F_A, E, ρ_A), where $F_A =_{df} (\mathsf{Au}\ A)$, and $\rho_A =_{df} \rho_{|(\mathsf{St}\ A)}$.

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 characterizes, 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 the remainder of this section we will deal with the UML semantics of HAs.

A *configuration* denotes a global state of a HA, composed of local states of component sequential automata:

Definition 74 (*Configurations*). A *configuration* of H is a set $\mathcal{C} \subseteq \bigcup_{A \in F} \sigma_A$ such that (i) $\exists_1 s \in \sigma_{A_{root}}.\ s \in \mathcal{C}$ and (ii) $\forall s, A.\ s \in \mathcal{C} \wedge A \in \rho\ s \Rightarrow \exists_1 s' \in A.\ s' \in \mathcal{C}$.

The set of all configurations of H is denoted by Conf_H, while $\mathcal{C}_{\mathrm{in}}$ denotes its *initial* configuration, namely the configuration composed only by initial states.

The operational semantics of a HA is defined as a LTS, where the states are the configuration/ input-queue pairs of the associated UMLSC and the transitions are characterized by the *step*-relation. Each transition of the LTS is labeled by the set of (unique identifiers of the) transitions of the associated UMLSC which have been fired in the step.

While in classical statecharts the external environment is modeled by a set, in the definition of UML statecharts, the nature of the input-queue of a statechart is not specified; in particular, the management policy of such a queue is not defined. In our overall approach to UMLSCs semantics definition, we choose *not* to fix any particular semantics, such as set, or multi-set or FIFO-queue etc., but to model the input queue in a policy-independent way, freely using a notion of abstract data types, as briefly described in Section 3. In addition to the operations described in that section, here we use also predicate $\mathsf{is\_join}^n_{j=1} \mathcal{D}_j\ \mathcal{I}$, which states that $\mathcal{I}$ is a possible *join* of $\mathcal{D}_1 \ldots \mathcal{D}_n$ and it is a way for expressing non-deterministic merge of $\mathcal{D}_1 \ldots \mathcal{D}_n$. Finally, 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).

Definition 75 (*Operational semantics*). The operational semantics of an HA $H = (F, E, \rho)$ is the LTS over $2^{\mathsf{Tr}\ H}$ $(\mathrm{Conf}_H \times \Theta_E, (\mathcal{C}_{\mathrm{in}}, \mathcal{E}_0), 2^{\mathsf{Tr}\ H}, \longrightarrow)$ where (i) $\mathrm{Conf}_H \times \Theta_E$ is the set of statuses, (ii) $(\mathcal{C}_{\mathrm{in}}, \mathcal{E}_0) \in \mathrm{Conf}_H \times \Theta_E$ is the initial status, with $\mathcal{C}_{\mathrm{in}}$ the configuration composed only of initial states of automata in F and $\mathcal{E}_0$ the given initial input queue,

Progress Rule

$$\frac{t \in LE_A\,\mathcal{C}\,\mathcal{E} \qquad \not\exists t' \in P \cup E_A\,\mathcal{C}\,\mathcal{E}.\ \pi\,t \sqsubset \pi\,t'}{A \uparrow P :: (\mathcal{C},\mathcal{E}) \xrightarrow{\{t\}} (DEST\ t, new(ACt))}$$

Stuttering Rule

$$\frac{\{s\} = \mathcal{C} \cap \sigma_A \qquad \rho_A\ s = \emptyset \qquad \forall t \in LE_A\,\mathcal{C}\,\mathcal{E}.\ \exists t' \in P.\ \pi\,t \sqsubset \pi\,t'}{A \uparrow P :: (\mathcal{C},\mathcal{E}) \xrightarrow{\emptyset} (\{s\}, \langle\rangle)}$$

Composition Rule

$$\frac{\begin{array}{l}\{s\} = \mathcal{C} \cap \sigma_A \\ \rho_A\ s = \{A_1,\ldots,A_n\} \neq \emptyset \\ \left(\bigwedge_{j=1}^{n} A_j \uparrow (P \cup LE_A\,\mathcal{C}\,\mathcal{E}) :: (\mathcal{C},\mathcal{E}) \xrightarrow{\mathcal{L}_j} (\mathcal{C}_j,\mathcal{E}_j)\right) \wedge \mathsf{is\_join}_{j=1}^{n}\,\mathcal{E}_j\,\mathcal{I} \\ \left(\bigcup_{j=1}^{n} \mathcal{L}_j = \emptyset\right) \Rightarrow (\forall t \in LE_A\,\mathcal{C}\,\mathcal{E}.\ \exists t' \in P.\ \pi\,t \sqsubset \pi\,t')\end{array}}{A \uparrow P :: (\mathcal{C},\mathcal{E}) \xrightarrow{\bigcup_{j=1}^{n} \mathcal{L}_j} (\{s\} \cup \bigcup_{j=1}^{n} \mathcal{C}_j, \mathcal{I})}$$

where

i) $LE_A\,\mathcal{C}\,\mathcal{E} =_{df}$
$\{t \in \delta_A \mid \{(SRC\,t)\} \cup (SR\,t) \subseteq \mathcal{C}\ \wedge (EV\,t) \in \mathcal{E} \wedge (\mathcal{C},\mathcal{E}) \models (G\,t)\}$;
$(\mathcal{C},\mathcal{E}) \models (G\,t)$ formalizes that guard $(G\,t)$ is true of $(\mathcal{C},\mathcal{E})$;

ii) $E_A\,\mathcal{C}\,\mathcal{E} =_{df} \bigcup_{A' \in (\mathsf{Au}\ A)} LE_{A'}\,\mathcal{C}\,\mathcal{E}$;

iii) $(DEST\ t) =_{df} \{s \mid \exists s' \in (TD\ t).\ (TGT\ t) \preceq s \preceq s'\}$;

iv) $\preceq$ is the state-nesting partial order and $\sqsubseteq$ is the priority partial order based on the priority mapping π: $\pi\,t \sqsubseteq \pi\,t'$ means that the priority of t is smaller than or equal to that of t'.

Fig. A.1. Core semantics of UML hierarchical automata.

(iii) $\longrightarrow \subseteq (\mathrm{Conf}_H \times \Theta_E) \times 2^{\mathsf{Tr}\ H} \times (\mathrm{Conf}_H \times \Theta_E)$, the *step*-transition relation, is the smallest relation which satisfies the rule given in Definition 3.

As usual, we write $(\mathcal{C},\mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}',\mathcal{E}')$ for $((\mathcal{C},\mathcal{E}),\mathcal{L},(\mathcal{C}',\mathcal{E}')) \in \longrightarrow$. Any such transition denotes the result of firing a maximal set $\mathcal{L}$ of non-conflicting transitions of the sequential automata of H which respect priorities. In the above-mentioned rule we make use of an auxiliary relation, namely $A \uparrow P :: (\mathcal{C},\mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}',\mathcal{E}')$. Such a relation, which is defined by the deduction system proposed in [24] and recalled in Fig. A.1, models labeled transitions of the semantics of HA A under specific constraints P related to transition priority. $\mathcal{L}$ 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 events) $\mathcal{C}'$ (resp. $\mathcal{E}'$).

The following lemma, proved in [31], gives some insights on the core semantics and will be used later in Appendix B.

Lemma 76. *For all HA* $H = (F, E, \rho), A \in F, P \subseteq (\mathit{Tr}\, H), \mathcal{E} \in \Theta_E, \mathcal{C} \in \mathrm{Conf}_H$, *s.t.* $\sigma_A \cap \mathcal{C} \neq \emptyset$ *the following holds*:

(i) $\exists \mathcal{L} \subseteq (\mathit{Tr}\, H), \mathcal{C}' \in \mathrm{Conf}_H, \mathcal{E}' \in \Theta_E.\ A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}', \mathcal{E}')$;

(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* $\mathcal{E}' = \langle\rangle$;

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

Appendix B. Detailed proofs

B.1. Proofs related to Section 3

Proof of Proposition 9. The direct implication holds trivially. So we prove that $\mathcal{M} \sim^{\mathsf{DetEx}} \mathcal{M}'$ implies $\mathcal{M} \sim \mathcal{M}'$. Suppose $\mathcal{M} \sim \mathcal{M}'$ does not hold. By definition of testing equivalence this means that there is an experimenter $\mathcal{T} = (T_U, \upsilon_{\mathrm{in}}, T_I, L_I, L_U, \rightarrow, \hookrightarrow)$ such that $\mathrm{Result}(\mathcal{T}, \mathcal{M}) \neq \mathrm{Result}(\mathcal{T}, \mathcal{M}')$ and suppose $\mathcal{T}$ is not input-deterministic. We know that there exists input-deterministic experimenter $\mathcal{T}' = (T_U', \upsilon_{\mathrm{in}}', T_I', L_I, L_U, \rightarrow', \hookrightarrow')$ and bijection $h_U : T_U' \mapsto T_U$ as in Lemma 80 below. Let us assume w.l.g. there exists $r \in \mathrm{Result}(\mathcal{T}, \mathcal{M}) \setminus \mathrm{Result}(\mathcal{T}, \mathcal{M}')$. There are two possibilities:

Case 1: $r = \top$

In this case, $\langle \mathcal{T}, \mathcal{M} \rangle$ has a successful computation

$$\upsilon_0 \,||\, s_0 \rightsquigarrow \upsilon_1 \,||\, s_1 \rightsquigarrow \ldots$$

while $< \mathcal{T}, \mathcal{M}' >$ has no successful computation. From Lemma 80 we know that $< \mathcal{T}', \mathcal{M} >$ has a successful computation, namely

$$(h_U^{-1}\, \upsilon_0) \,||\, s_0 \rightsquigarrow (h_U^{-1}\, \upsilon_1) \,||\, s_1 \rightsquigarrow \ldots$$

Since $\mathcal{M} \sim^{\mathsf{DetEx}} \mathcal{M}'$ by hypothesis, also $< \mathcal{T}', \mathcal{M}' >$ must have a successful computation

$$\upsilon_0' \,||\, s_0' \rightsquigarrow \upsilon_1' \,||\, s_1' \rightsquigarrow \ldots$$

But then, again from Lemma 80 below,

$$(h_U\, \upsilon_0') \,||\, s_0 \rightsquigarrow (h_U\, \upsilon_1') \,||\, s_1 \rightsquigarrow \ldots$$

is a successful computation of $< \mathcal{T}, \mathcal{M}' >$, which is a contradiction since $\top \notin \mathrm{Result}(\mathcal{T}, \mathcal{M}')$.

Case 2: $r = \bot$

In this case, $\langle \mathcal{T}, \mathcal{M} \rangle$ has an unsuccessful computation

$$\upsilon_0 \,||\, s_0 \rightsquigarrow \upsilon_1 \,||\, s_1 \rightsquigarrow \ldots$$

while $< \mathcal{T}, \mathcal{M}' >$ has no unsuccessful computation. From Lemma 80 we know that $\langle \mathcal{T}', \mathcal{M} \rangle$ has an unsuccessful computation, namely

$$(h_U^{-1}\, \upsilon_0) \mid\mid s_0 \rightsquigarrow (h_U^{-1}\, \upsilon_1) \mid\mid s_1 \rightsquigarrow \ldots$$

Since $\mathcal{M} \sim^{\mathsf{DetEx}} \mathcal{M}'$ by hypothesis, also $\langle \mathcal{T}', \mathcal{M}' \rangle$ must have an unsuccessful computation

$$\upsilon_0' \mid\mid s_0' \rightsquigarrow \upsilon_1' \mid\mid s_1' \rightsquigarrow \ldots$$

But then, again from Lemma 80 below,

$$(h_U\, \upsilon_0') \mid\mid s_0 \rightsquigarrow (h_U\, \upsilon_1') \mid\mid s_1 \rightsquigarrow \ldots$$

is an unsuccessful computation of $\langle \mathcal{T}, \mathcal{M}' \rangle$, which is a contradiction since $\perp \notin \text{Result}(\mathcal{T}, \mathcal{M}')$. $\square$

In order to prove Lemma 80, we first need some auxiliary notation.

Definition 77. For experimenter $(T_U, \upsilon_{\text{in}}, T_I, L_I, L_U, \rightarrow, \hookrightarrow)$ and all $\iota \in T_I$, let Q_ι, D_ι, MD_ι, and MD be the following sets:

$$Q_\iota =_{df} \{(\iota, u, \upsilon) \mid \iota \overset{u}{\hookrightarrow} \upsilon\}$$

$$D_\iota =_{df} \{q \subseteq Q_\iota \mid \forall u, \upsilon_1, \upsilon_2.\ ((\iota, u, \upsilon_1) \in q \wedge (\iota, u, \upsilon_2) \in q) \Rightarrow \upsilon_1 = \upsilon_2\}$$

$$MD_\iota =_{df} \{d \in D_\iota \mid \nexists d' \in D_\iota.\ d \subset d'\}$$

$$MD =_{df} \bigcup_{\iota \in T_I} MD_\iota$$

We illustrate the above definition by the following example.

Example 78. Let us consider the experimenter shown graphically in Fig. B.1, where L_I is the set $\{i1, i2\}$, L_U is the set $\{u1, u2, u3\}$, and, for the sake of readability, input states are shown as triangles and output states are shown as circles. We get

$$Q_{\iota_1} = \{(\iota_1, u1, \upsilon_5), (\iota_1, u1, \upsilon_6), (\iota_1, u2, \upsilon_4), (\iota_1, u2, \upsilon_5), (\iota_1, u2, \upsilon_6), (\iota_1, u3, \upsilon_4)\}$$

and

$$Q_{\iota_2}\{(\iota_2, u1, \upsilon_6), (\iota_2, u2, \upsilon_5), (\iota_2, u3, \upsilon_4)\}$$

i.e. Q_ι is the subset of $\hookrightarrow$ obtained by selection over the first element, which is required to be ι.

$$MD_{\iota_1} = \{d_{1a}, d_{1b}, d_{1c}, d_{1d}, d_{1e}, d_{1f}\}$$

where

$$\begin{aligned}
d_{1a} &= \{(\iota_1, u1, \upsilon_5), (\iota_1, u2, \upsilon_4), (\iota_1, u3, \upsilon_4)\}\\
d_{1b} &= \{(\iota_1, u1, \upsilon_6), (\iota_1, u2, \upsilon_4), (\iota_1, u3, \upsilon_4)\}\\
d_{1c} &= \{(\iota_1, u1, \upsilon_5), (\iota_1, u2, \upsilon_5), (\iota_1, u3, \upsilon_4)\}\\
d_{1d} &= \{(\iota_1, u1, \upsilon_6), (\iota_1, u2, \upsilon_5), (\iota_1, u3, \upsilon_4)\}\\
d_{1e} &= \{(\iota_1, u1, \upsilon_5), (\iota_1, u2, \upsilon_6), (\iota_1, u3, \upsilon_4)\}\\
d_{1f} &= \{(\iota_1, u1, \upsilon_6), (\iota_1, u2, \upsilon_6), (\iota_1, u3, \upsilon_4)\}
\end{aligned}$$

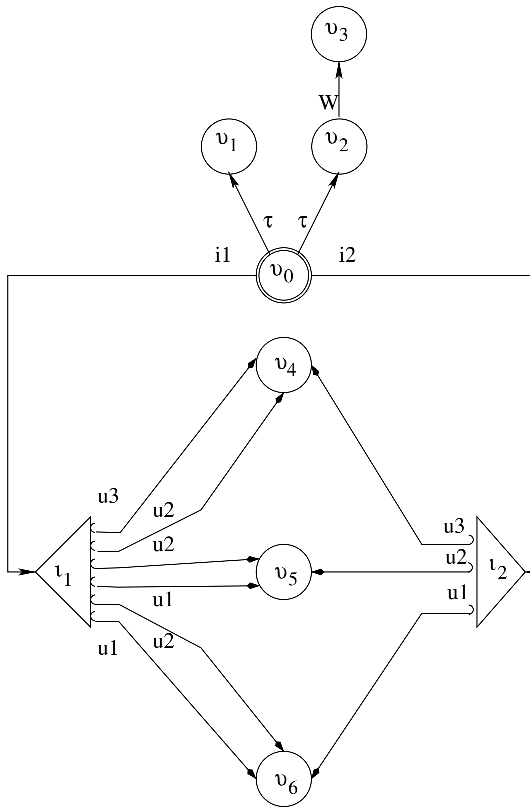


Fig. B.1. A sample experimenter.

and

$$MD_{\iota_2} = \{d_{2a}\}$$

where

$$d_{2a} = \{(\iota_2, u1, \upsilon_6), (\iota_2, u2, \upsilon_5), (\iota_2, u3, \upsilon_4)\}$$

i.e. MD_ι is the set of all *deterministic* subsets of Q_ι which are *maximal*.[9]

In Fig. B.2 a procedure is shown which, given experimenter $\mathcal{T}$ can be used for building another experimenter $\mathcal{T}'$, which is input-deterministic and such that there is a precise correspondence between the computations which the two experimenters give rise to when experimenting with any LTS, as established by Lemma 80 below. Notice that $\mathcal{T}'$ is defined only up to isomorphisms.

Example 79. Fig. B.3 shows an input-deterministic experimenter obtained by the application of the procedure of Fig B.2 to the experimenter of Fig. B.1. Bijection h_U maps

[9] Conceptually, D_ι and MD_ι are essentially the extension of functions mf and $func$—defined in Section 4—to ternary relations.

Given experimenter $\mathcal{T} = (T_U, \upsilon_{\text{in}}, T_I, L_I, L_U, \rightarrow, \hookrightarrow)$, with reference to Def. 77, let experimenter $\mathcal{T}' = (T'_U, \upsilon'_{\text{in}}, T'_I, L_I, L_U, \rightarrow', \hookrightarrow')$ be as follows:

(1) T'_U is a set isomorphic to T_U, where $h_U : T'_U \mapsto T_U$ is the relevant bijection;

(2) $\upsilon'_{\text{in}} = (h_U\, \upsilon_{\text{in}})$;

(3) T'_I is a set isomorphic to MD, where $h_I : T'_I \mapsto MD$ is the relevant bijection;

(4) For all $\upsilon'_1, \upsilon'_2 \in T'_U, \mu \in \{\tau, \mathbf{W}\}, \iota' \in T'_I, i \in L_I$:

i) $\upsilon'_1 \xrightarrow{\mu}{}' \upsilon'_2$ iff $(h_U\, \upsilon'_1) \xrightarrow{\mu} (h_U\, \upsilon'_2)$;

ii) $\upsilon'_1 \xrightarrow{i}{}' \iota'$ iff there exists $\iota \in T_I$ such that $(h_U\, \upsilon'_1) \xrightarrow{i} \iota$ and $(h_I\, \iota') \in MD_\iota$;

(5) For all $\iota' \in T'_I$ there exists $\iota \in T_I$ such that:

$\{(u, \upsilon') \mid \iota' \overset{u}{\hookrightarrow}{}' \upsilon'\} = \{(u, \upsilon') \mid (\iota, u, (h_U\, \upsilon')) \in (h_I\, \iota')\}$

Fig. B.2. Procedure for building input-deterministic experimenter.

each output state υ_{ja} to υ_j, for $j = 0, \ldots, 6$ while h_I maps input state ι_{1x} to d_{1x}, for $x \in \{a, b, c, d, e, f\}$ and input state ι_{2a} to d_{2a}. The transition relations are built according to rules (4) and (5) of the procedure of Fig B.2. The basic idea behind the procedure of Fig B.2 is to 'push' the non-determinism arising from the input transitions outgoing from an input state ι 'backwards' to the output transitions incoming to ι, making as many 'copies' of this state and related incoming transitions as the number of distinct maximal deterministic sets which the input transitions give rise to. An intuitive justification of the correctness of the procedure can be found by keeping in mind the standard result of testing equivalence according to which the LTSs of Fig. B.4(a) and (b) have the same deadlock capabilities—indeed they are testing equivalent LTSs—and the LTS of Fig. B.4(b) can be seen as obtained from that of Fig. B.4(a) by means of 'pushing' non-determinism from the transitions outgoing from s_1 'backwards' to those incoming to s_1, while making two 'copies', s'_1 and s''_1, of that state.

Lemma 80. *For each experimenter $\mathcal{T} = (T_U, \upsilon_{in}, T_I, L_I, L_U, \rightarrow, \hookrightarrow)$ there exists an input-deterministic experimenter $\mathcal{T}' = (T'_U, \upsilon'_{in}, T'_I, L_I, L_U, \rightarrow', \hookrightarrow')$ and bijection $h_U : T'_U \mapsto T_U$ such that, for each LTS $\mathcal{M} = (S, s_{in}, L_I \times L_U, \rightarrow)$*

$$\eta = \upsilon_0 \,||\, s_0 \rightsquigarrow \upsilon_1 \,||\, s_1 \rightsquigarrow \ldots$$

is a computation of $\langle \mathcal{T}, \mathcal{M} \rangle$ if and only if

$$\eta' = (h_U^{-1}\, \upsilon_0) \,||\, s_0 \rightsquigarrow (h_U^{-1}\, \upsilon_1) \,||\, s_1 \rightsquigarrow \ldots$$

is a computation of $< \mathcal{T}', \mathcal{M} >$, and η is successful iff η' is successful.

Proof. Let $\mathcal{T}'$ be an experimenter built from $\mathcal{T}$ according to the procedure shown in Fig. B.2. We note that, for all $\iota' \in T'_I$, maximality of $(h_I\, \iota')$—which follows from the definition of MD—implies in turn that for all $u \in L_U$ there exists $\upsilon' \in T'_U$ such that $\iota' \overset{u}{\hookrightarrow}{}' \upsilon'$. Thus $\hookrightarrow'$ is total. Moreover, $\hookrightarrow'$ is deterministic by construction since every element of MD is

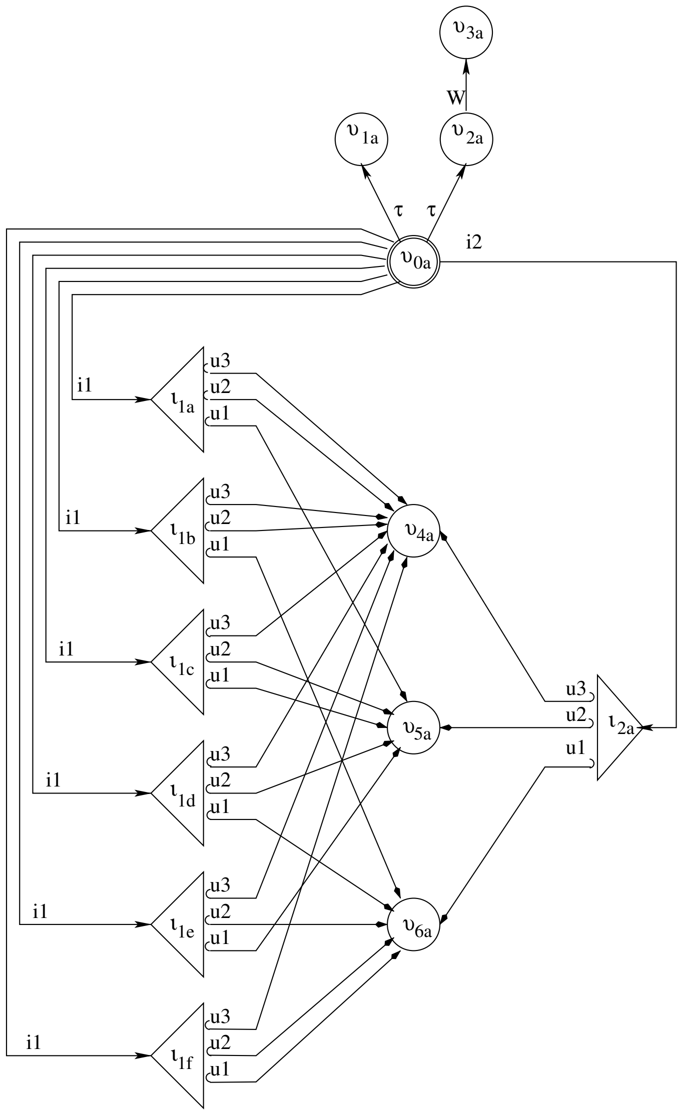


Fig. B.3. Input-deterministic experimenter $\mathcal{T}'$ for $\mathcal{T}$ of Fig. B.1.

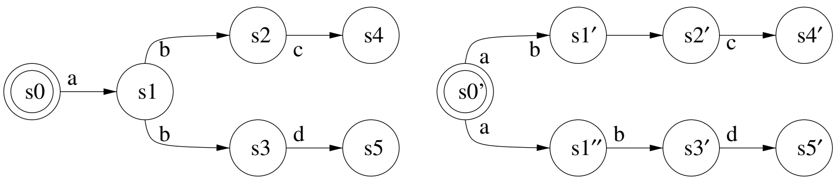
deterministic by definition and $(h_I\ \iota') \in MD$. We can then conclude that $\mathcal{T}'$ is an input-deterministic experimenter.

Now we show that for all $\upsilon_1, \upsilon_2 \in T_U$ and $s_1, s_2 \in S$, if $\upsilon_1 \mid\mid s_1 \rightsquigarrow \upsilon_2 \mid\mid s_2$ in the experimental system $\langle \mathcal{T}, \mathcal{M} \rangle$, then $(h_U^{-1}\ \upsilon_1) \mid\mid s_1 \rightsquigarrow (h_U^{-1}\ \upsilon_2) \mid\mid s_2$ in the experimental system $< \mathcal{T}', \mathcal{M} >$. According to Definition 5 there are two possibilities:

Case 1: $\upsilon_1 \xrightarrow{\tau} \upsilon_2$ *and* $s_1 = s_2$.

By point (4.i) of the definition of $\mathcal{T}'$ we get that also $(h_U^{-1}\ \upsilon_1) \xrightarrow{\tau}{}' (h_U^{-1}\ \upsilon_2)$, and by Definition 5 we get $(h_U^{-1}\ \upsilon_1) \mid\mid s_1 \rightsquigarrow (h_U^{-1}\ \upsilon_2) \mid\mid s_1$.

Case 2: *there exist* $i \in L_I, u \in L_U, \iota \in T_I$ *such that* $\upsilon_1 \xrightarrow{i} \iota, \iota \overset{u}{\hookrightarrow} \upsilon_2$ *and* $s_1 \xrightarrow{(i,u)} s_2$.



(a) (b)

Fig. B.4. Two testing equivalent LTSs.

Since $\iota \overset{u}{\hookrightarrow} \upsilon_2$, we know, by definition of MD_ι, that there exists $d \in MD_\iota$ such that $(\iota, u, \upsilon_2) \in d$. By points (1) and (3) of the definition of $\mathcal{T}'$ we know that $(h_U^{-1}\, \upsilon_1) \in T'_U$ and $(h_I^{-1}\, d) \in T'_I$. So, by point (4.ii) we get $(h_U^{-1}\, \upsilon_1) \xrightarrow{i'} (h_I^{-1}\, d)$. Moreover, by point (5) of the definition of $\mathcal{T}'$ we also get $(h_I^{-1}\, d) \overset{u}{\hookrightarrow}{}' (h_U^{-1}\, \upsilon_2)$ since $(\iota, u, \upsilon_2) \in d \in MD_\iota$. Finally, we get $(h_U^{-1}\, \upsilon_1) \,||\, s_1 \rightsquigarrow (h_U^{-1}\, \upsilon_2) \,||\, s_2$ by Definition 5.

Similarly, we show the converse, i.e. that for all $\upsilon'_1, \upsilon'_2 \in T'_U$ and $s_1, s_2 \in S$, if $\upsilon'_1 \,||\, s_1 \rightsquigarrow \upsilon'_2 \,||\, s_2$ in the experimental system $< \mathcal{T}', \mathcal{M} >$, then $(h_U\, \upsilon'_1) \,||\, s_1 \rightsquigarrow (h_U\, \upsilon'_2) \,||\, s_2$ in the experimental system $\langle \mathcal{T}, \mathcal{M} \rangle$.

Again there are two cases according to Definition 5:

Case 1: $\upsilon'_1 \xrightarrow{\tau}{}' \upsilon'_2$ *and* $s_1 = s_2$.

By point (4.i) of the definition of $\mathcal{T}'$ we get that also $(h_U\, \upsilon'_1) \xrightarrow{\tau} (h_U\, \upsilon'_2)$, and by Definition 5 we get $(h_U\, \upsilon'_1) \,||\, s_1 \rightsquigarrow (h_U\, \upsilon'_2) \,||\, s_1$.

Case 2: *there exist* $i \in L_I, u \in L_U, \iota' \in T'_I$ *such that* $\upsilon'_1 \xrightarrow{i'} \iota', \iota' \overset{u}{\hookrightarrow}{}' \upsilon'_2$ *and* $s_1 \xrightarrow{(i,u)} s_2$.

By point (4.ii) of the definition of $\mathcal{T}'$ there exists $\iota \in T_I$ such that $(h_U\, \upsilon'_1) \xrightarrow{i} \iota$ and $(h_I\, \iota') \in MD_\iota$. Moreover, by point (5) of the definition of $\mathcal{T}'$, there exists $\bar{\iota} \in T_I$ such that $(\bar{\iota}, u, (h_U\, \upsilon'_2)) \in (h_I\, \iota')$. But $(h_I\, \iota') \in MD_\iota$ and $(\bar{\iota}, u, (h_U\, \upsilon'_2)) \in (h_I\, \iota')$ imply $\bar{\iota} = \iota$ and $(\iota, u, (h_U\, \upsilon'_2)) \in Q_\iota$. Thus we have also $\iota \overset{u}{\hookrightarrow} (h_U\, \upsilon'_2)$, and by Definition 5 we get $(h_U\, \upsilon'_1) \,||\, s_1 \rightsquigarrow (h_U\, \upsilon'_2) \,||\, s_2$.

The correspondence between the computations of $\langle \mathcal{T}, \mathcal{M} \rangle$ and $\langle \mathcal{T}', \mathcal{M} \rangle$, i.e.

$$\eta = \upsilon_0 \,||\, s_0 \rightsquigarrow \upsilon_1 \,||\, s_1 \rightsquigarrow \ldots$$

is a computation of $\langle \mathcal{T}, \mathcal{M} \rangle$ if and only if

$$\eta' = (h_U^{-1}\, \upsilon_0) \,||\, s_0 \rightsquigarrow (h_U^{-1}\, \upsilon_1) \,||\, s_1 \rightsquigarrow \ldots$$

is a computation of $\langle \mathcal{T}', \mathcal{M} \rangle$ is a direct consequence of the above property of $\mathcal{T}'$; moreover, the fact that η is successful if and only if η' is successful follows directly from the definition of successful computation and from point (4.i) of the definition of $\mathcal{T}'$. □

B.2. Proofs related to Section 4

Proof of Proposition 14

$$\exists \mathcal{L}.\ \mathcal{L} \neq \emptyset \ \wedge\ (\mathcal{C}, \mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}', (join\ \mathcal{E}''\ \mathcal{E}'))$$

$\Leftrightarrow$ {Def. of $\xrightarrow{\mathcal{L}}$ (see Def. 3)}

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

$\Leftrightarrow$ {Def. 13, Def. of $Queue(\mathcal{F})$}

$\exists e \in E.\ \mathcal{C} \xrightarrow{(e,\mathcal{E}')} \mathcal{C}' \wedge Queue(\mathcal{E}) \xrightarrow{e} \lambda X.Queue(join\ \mathcal{E}''\ X)$

$\Leftrightarrow$ {Def. 5, $Queue(\mathcal{F})$ does not perform silent moves}

$(Queue(\mathcal{E})\ ||\ \mathcal{C}) \rightsquigarrow (Queue(join\ \mathcal{E}''\ \mathcal{E}')\ ||\ \mathcal{C}')$ □

Proof of Proposition 16. We first prove that $\mathcal{M} \sim \mathcal{M}'$ implies $\mathcal{M} \eqsim \mathcal{M}'$. By contradiction, suppose $\mathcal{M} \sim \mathcal{M}'$ and that $\mathcal{M} \eqsim \mathcal{M}'$ does *not* hold. W.l.g. suppose $\mathcal{M} \sqsubseteq\!\!\!\sim \mathcal{M}'$ does not hold; this can only happen if one of the two statements does not hold: $\mathcal{M} \sqsubset\!\!\!\sim_{MAY} \mathcal{M}'$ or $\mathcal{M} \sqsubset\!\!\!\sim_{MUST} \mathcal{M}'$. In the first case, there would exist a test $\mathcal{T}$ for which $\top \in \text{Result}(\mathcal{T}, \mathcal{M}) \setminus \text{Result}(\mathcal{T}, \mathcal{M}')$, which would contradict $\mathcal{M} \sim \mathcal{M}'$. In the second case, there would exist a test $\mathcal{T}$ for which $\bot \in \text{Result}(\mathcal{T}, \mathcal{M}') \setminus \text{Result}(\mathcal{T}, \mathcal{M})$, also contradicting $\mathcal{M} \sim \mathcal{M}'$.

We now prove that $\mathcal{M} \eqsim \mathcal{M}'$ implies $\mathcal{M} \sim \mathcal{M}'$. Again by contradiction, suppose $\mathcal{M} \eqsim \mathcal{M}'$ and that $\mathcal{M} \sim \mathcal{M}'$ does *not* hold. If $\mathcal{M} \not\sim \mathcal{M}'$, then there exists a test $\mathcal{T}$ such that $\text{Result}(\mathcal{T}, \mathcal{M}) \neq \text{Result}(\mathcal{T}, \mathcal{M}')$. Suppose, w.l.g., that there exists $r \in \text{Result}(\mathcal{T}, \mathcal{M}') \setminus \text{Result}(\mathcal{T}, \mathcal{M})$. Again there are two cases. If $\top \in \text{Result}(\mathcal{T}, \mathcal{M}') \setminus \text{Result}(\mathcal{T}, \mathcal{M})$, then $\mathcal{M}' \sqsubset\!\!\!\sim_{MAY} \mathcal{M}$ would be violated, contradicting $\mathcal{M} \eqsim \mathcal{M}'$. If instead $\bot \in \text{Result}(\mathcal{T}, \mathcal{M}') \setminus \text{Result}(\mathcal{T}, \mathcal{M})$, then $\mathcal{M} \sqsubset\!\!\!\sim_{MUST} \mathcal{M}'$ would be violated, again contradicting $\mathcal{M} \eqsim \mathcal{M}'$. □

Proof of Theorem 21. In order to prove Theorem 21, we first need some auxiliary notions and results. First of all, we use the experimenter $\mathcal{W}$ which always experiences *success*, i.e. $\mathcal{W} \triangleq \mathbf{W}; \delta$. We now define a particular class of experimenters. The first kind of such experimenters is that of those which can fail only if the LTS they are experimenting with, after having performed the input/output sequence $\gamma = (i_1, u_1)(i_2, u_2) \ldots (i_n, u_n)$, will react with output u on input i. The definition of such an experimenter $(Ex(\gamma, (i, u)), L_U)$ is given in Fig. B.5. The second kind of experimenters of interest is that of those which can fail only if the LTS under test, after having performed the input/output sequence γ as before cannot accept any input from finite set $I = \{i'_1, \ldots i'_k\}$. The definition of such an experimenter $(Ex(\gamma, I), L_U)$ is given in Fig. B.6. We let EX denote the set of all the experimenters of the kind $Ex(\gamma, (i, u))$ or $Ex(\gamma, I)$. Moreover, in the sequel, for $A \subseteq L$ we let $(IN\ A)$ denote the set $\{i \mid (i, u) \in A\}$.

Finally, a third kind of experimenters is used, $(Ex(\gamma), L_U)$, which succeeds only if the system under test can perform γ, for $\gamma = (i_1, u_1) \ldots (i_n, u_n)$. The definition for $Ex(\gamma)$ shown in Fig. B.7.

The proof of Theorem 21 follows:

Part (a)

We first show that if $\mathcal{M} \sqsubset\!\!\!\sim_{MAY} \mathcal{M}'$ then $\mathcal{M} <<_{MAY} \mathcal{M}'$. Suppose $\mathcal{M} <<_{MAY} \mathcal{M}'$ does not hold; then we can set up the following derivation,

$\mathcal{M} <<_{MAY} \mathcal{M}' \equiv \text{FALSE}$

$\Rightarrow$ {Def. of $<<_{MAY}$}

$\mathsf{lan}\ \mathcal{M} \not\subseteq \mathsf{lan}\ \mathcal{M}'$

$$
\begin{array}{l}
Ex((i_1,u_1)\ldots(i_n,u_n):L^*,(i,u):L) \triangleq \\
\tau;\mathcal{W} \\
+ \\
i_1;\lambda x.(x \neq u_1 \Rightarrow \tau;\mathcal{W} \\
\hspace{2em} + \\
\hspace{2em} x = u_1 \Rightarrow (\tau;\mathcal{W} \\
\hspace{6em} + \\
\hspace{6em} i_2;\lambda x.(x \neq u_2 \Rightarrow \tau;\mathcal{W} \\
\hspace{8em} + \\
\hspace{8em} x = u_2 \Rightarrow \\
\hspace{8em} \ddots \\
\hspace{13em} (\tau;\mathcal{W} \\
\hspace{13em} + \\
\hspace{13em} i_n;\lambda x.(x \neq u_n \Rightarrow \tau;\mathcal{W} \\
\hspace{15em} + \\
\hspace{15em} x = u_n \Rightarrow (\tau;\mathcal{W} \\
\hspace{19em} + \\
\hspace{19em} i;\lambda x.(x \neq u \Rightarrow \tau;\mathcal{W} \\
\hspace{21em} + \\
\hspace{21em} x = u \Rightarrow \delta \\
\hspace{19em}) \hspace{2em}) \\
\hspace{13em}) \hspace{2em}) \\
\hspace{8em} \iddots \\
\hspace{6em}) \hspace{2em}) \\
\hspace{2em})
\end{array}
$$

Fig. B.5. Definition of $Ex((i_1,u_1)\ldots(i_n,u_n),(i,u))$.

$\Rightarrow$ {Set theory}

$\exists\gamma \in \mathsf{lan}\,\mathcal{M}.\ \gamma \notin \mathsf{lan}\,\mathcal{M}'$

$\Rightarrow$ {Def. of $Ex(\gamma)$, Def. 7}

$\top \in \text{Result}(Ex(\gamma),\mathcal{M}) \wedge \top \notin \text{Result}(Ex(\gamma),\mathcal{M}')$

thus concluding that $\mathcal{M} \sqsubseteq_{MAY} \mathcal{M}'$ does not hold.

Now we show that if $\mathcal{M} \ll_{MAY} \mathcal{M}'$ then $\mathcal{M} \sqsubseteq_{MAY} \mathcal{M}'$. Suppose $\top \in \text{Result}(\mathcal{T},\mathcal{M})$ for some experimenter $\mathcal{T}$, so that there must be a computation starting from $\upsilon_{\text{in}} \mid\mid s_{\text{in}}$ leading to success and there must be a finite prefix of the computation, say

$$\upsilon_{\text{in}} \mid\mid s_{\text{in}} \rightsquigarrow \ldots \upsilon_k \mid\mid s$$

which leads to success. Such a prefix gives rise to a derivation $s_{\text{in}} \xrightarrow{\gamma} s$, on the side of $\mathcal{M}$, for $\gamma = (i_1,u_1)\ldots(i_n,u_n)$, and to a sequence of output transitions $\upsilon_j \xrightarrow{\mu_j} \mathcal{O}_j$, for $j = 0\ldots k-1$ such that either $\mu_j = i_i$ and $\mathcal{O}_j \overset{u_i}{\hookrightarrow} \upsilon_{j+1}$, for some i with $1 \leq i \leq n$, or

$$
\begin{array}{l}
Ex((i_1,u_1)\ldots(i_n,u_n):L^*,\{i'_1,\ldots i'_k\}:2^{L_I}) \triangleq \\
\tau;\mathcal{W} \\
+ \\
i_1;\lambda x.(x\neq u_1 \Rightarrow \tau;\mathcal{W} \\
\quad\quad + \\
\quad\quad x=u_1 \Rightarrow(\tau;\mathcal{W} \\
\quad\quad\quad\quad + \\
\quad\quad\quad\quad i_2;\lambda x.(x\neq u_2 \Rightarrow \tau;\mathcal{W} \\
\quad\quad\quad\quad\quad\quad + \\
\quad\quad\quad\quad\quad\quad x=u_2 \Rightarrow \\
\quad\quad\quad\quad\quad\quad \ddots \\
\quad\quad\quad\quad\quad\quad\quad\quad (\tau;\mathcal{W} \\
\quad\quad\quad\quad\quad\quad\quad\quad + \\
\quad\quad\quad\quad\quad\quad\quad\quad i_n;\lambda x.(x\neq u_n \Rightarrow \tau;\mathcal{W} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad + \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad x=u_n \Rightarrow((i'_1;\lambda x.\tau;\mathcal{W}) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad + \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \vdots \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad + \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad (i'_k;\lambda x.\tau;\mathcal{W})) \\
\quad\quad\quad\quad\quad\quad\quad\quad)\quad) \\
\quad\quad\quad\quad\quad\quad \iddots \\
\quad\quad\quad\quad)\quad) \\
\quad\quad)
\end{array}
$$

Fig. B.6. Definition of $Ex((i_1,u_1)\ldots(i_n,u_n),\{i'_1,\ldots i'_k\})$.

$\mu_j = \tau$ and $\upsilon_{j+1} = \mathcal{O}_j$. Notice that the derivation on the side of the experimenter involves a sequence γ' which is equal to γ up to τ moves. We know that $\mathsf{lan}\ \mathcal{M} \subseteq \mathsf{lan}\ \mathcal{M}'$ since $\mathcal{M} \ll_{MAY} \mathcal{M}'$; so also $\mathcal{M}'$ can perform $s'_{\text{in}} \xrightarrow{\gamma} s'$ for some s', and thus can be composed with the derivation we had for $\mathcal{T}$. And since this experimenter reported success somewhere in such a derivation, also this time it will do so. If the sequence we built is not maximal, we can extend it with further derivations starting from $\upsilon_k \,||\, s'$. In any case we found a successful computation for $\mathcal{T} \,||\, \mathcal{M}'$. This proves $\mathcal{M} \sqsubseteq_{MAY} \mathcal{M}'$.

Part (b)

The implication $\mathcal{M} \sqsubseteq_{MUST} \mathcal{M}' \Rightarrow \mathcal{M} \ll_{MUST} \mathcal{M}'$ follows from Lemma 82 below. The converse can be proved as follows. Suppose $\mathcal{M} \ll_{MUST} \mathcal{M}'$ and $\perp \notin \text{Result}(\mathcal{T},\mathcal{M})$. Let us consider an arbitrary computation starting from $\mathcal{T} \,||\, \mathcal{M}'$:

$$\upsilon_{\text{in}} \,||\, s'_{\text{in}} \rightsquigarrow \ldots \upsilon_k \,||\, s' \ldots \tag{B.1}$$

There are two possibilities: either the above sequence is finite, or it is infinite. Let us first consider the case in which it is finite and stops at $\upsilon_k \,||\, s'$. We must show that for

$$
\begin{array}{l}
Ex((i_1,u_1)\ldots(i_n,u_n) : L^*) \triangleq \\
i_1; \lambda x.(x \neq u_1 \Rightarrow \delta \\
\quad + \\
\quad x = u_1 \Rightarrow i_2; \lambda x.(x \neq u_2 \Rightarrow \delta \\
\qquad\qquad + \\
\qquad\qquad x = u_2 \Rightarrow \\
\qquad\qquad \ddots \\
\qquad\qquad\qquad i_n; \lambda x.(x \neq u_n \Rightarrow \delta \\
\qquad\qquad\qquad\quad + \\
\qquad\qquad\qquad\quad x = u_n \Rightarrow \mathcal{W} \\
\qquad\qquad\qquad\quad) \\
\qquad\qquad \cdot^{\cdot^{\cdot}} \\
\qquad\qquad) \\
\quad)
\end{array}
$$

Fig. B.7. Definition of $Ex((i_1,u_1)\ldots(i_n,u_n))$.

some $i, 0 \leq i \leq k$, $\upsilon_i \in$ *Success*. This computation gives rise to two derivations: one on the side of $\mathcal{M}'$, $s'_{\text{in}} \xrightarrow{\gamma} s'$, for some $\gamma \in L^*$, and one on the side of the experimenter, starting with υ_{in}, ending with υ_k, and involving γ' which is equal to γ up to occurrences of τ. Now, $(\mathsf{S}\ s'\ \epsilon) \in (\mathsf{AS}\ \mathcal{M}'\ \gamma)$ and then there exists $T \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma))$ with $T \subseteq (\mathsf{S}\ s'\ \epsilon)$. Moreover, since $mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma)) \subset\subset mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$ we can find $Z' \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$ such that $Z' \subseteq T$. Thus we get $Z' \subseteq (\mathsf{S}\ s'\epsilon)$. Now there are three cases for Z':

(i) $Z' \in \mathsf{AS}\ \mathcal{M}\ \gamma$. In this case there exists a s such that $s_{\text{in}} \xrightarrow{\gamma} s$ and $Z' = (\mathsf{S}\ s\ \epsilon) \subseteq (\mathsf{S}\ s'\ \epsilon)$. So $\upsilon_k \mid\mid s$ cannot be extended and therefore the derivation $s_{\text{in}} \xrightarrow{\gamma} s$ can be combined with the above-mentioned derivation for $\mathcal{T}$ involving γ', in order to give a computation $\upsilon_{\text{in}} \mid\mid s_{\text{in}} \rightsquigarrow \ldots \upsilon_k \mid\mid s$ for $\mathcal{T} \mid\mid \mathcal{M}$. Since $\perp \notin \text{Result}(\mathcal{T}, \mathcal{M})$, there must be $i, 0 \leq i \leq k$, such that $\upsilon_i \in$ *Success*.

(ii) $Z' \in \mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma) \setminus \mathsf{AS}\ \mathcal{M}\ \gamma$. In this case there exists s such that $s_{\text{in}} \xrightarrow{\gamma} s$ and $(\mathsf{S}\ s\ \epsilon) \in \mathsf{AS}\ \mathcal{M}\ \gamma$ with $(\mathsf{S}\ s\ \epsilon) \subseteq Z'$. Thus we have that $(\mathsf{S}\ s\ \epsilon) \subseteq (\mathsf{S}\ s'\ \epsilon)$. Then a similar reasoning as in case i) can be applied.

(iii) $Z' \in (mfs\ (\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))) \setminus \mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)$. In this case, there exists a set K in $\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)$ such that $Z' \in mfs\{K\}$. Moreover there exists a s such that $s_{\text{in}} \xrightarrow{\gamma} s$ and $\mathsf{S}\ s\ \epsilon \in \mathsf{AS}\ \mathcal{M}\ \gamma$, such that $\mathsf{S}\ s\ \epsilon \subseteq K$. We know that $Z' \subseteq \mathsf{S}\ s'\ \epsilon$. We know also that $\upsilon_k \mid\mid s'$ cannot be extended, and that $IN\ (\mathsf{S}\ s\ \epsilon) \subseteq INZ'$ because of the definition of mfs. All this together brings to the fact that $\upsilon_k \mid\mid s$ cannot be extended. Then a similar reasoning as in case i) can be applied.

Let us now consider the case in which the computation is infinite. Also in this case the computation gives rise to two derivations, which may be infinite: one on the side of $\mathcal{M}'$, $s'_{\text{in}} \xrightarrow{\gamma}$, for some γ, and one on the side of the experimenter, starting with υ_{in}, involving γ' which is equal to γ up to occurrences of τ. Both γ and γ' may be infinite sequences. Suppose now that for every natural number n there exists $m > n$ such that the m-th element of γ' is different from τ. This means that for each finite prefix $\bar{\gamma}$ of γ there exists s' such that $(\mathsf{S}\ s'\ \epsilon) \in \mathsf{AS}\ \mathcal{M}'\ \bar{\gamma}$ and so there exists $T \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \bar{\gamma}))$ with $T \subseteq (\mathsf{S}\ \mathcal{M}'\ \epsilon)$. But then, since $mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \bar{\gamma})) \subset\subset mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \bar{\gamma}))$, we can find $Y \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \bar{\gamma}))$ with $Y \subseteq T$.This means that $\mathsf{AS}\ \mathcal{M}\ \bar{\gamma}$ is non-empty and then there exists s such that $s_{\text{in}} \xrightarrow{\bar{\gamma}} s$. But then, since the above holds for every finite prefix of γ, we get also $s_{\text{in}} \xrightarrow{\gamma}$ and we can build an infinite computation by composing the derivation with the above derivation on the side of the experimenter involving γ'. Since $\perp \notin \text{Result}(\mathcal{T}, \mathcal{M})$, there will be a $\upsilon_i \in \mathit{Success}$ for some i. So the computation (B.1) above is successful. A similar reasoning applies also to the case in which there exists n such that the m-th element of γ' is τ for all $m \geq n$. Both in the case the successful state υ_i occurs in the silent suffix of γ' and in the case in which it occurs before such a suffix we can build a successful computation proceeding as above. Thus we conclude that in all cases $\perp \notin \text{Result}(\mathcal{T}, \mathcal{M}')$, and so $\mathcal{M} \sqsubseteq_{MUST} \mathcal{M}'$.

Part (c)
Obviously follows from parts (a) and (b). □

The above proof used Lemma 82 below, which in turn uses the the following lemma, which shows the relationship between $\sqsubseteq^{EX}_{MUST}$ and the languages of the relevant LTSs. Lemma 82 shows that EX is sufficiently expressive for the $MUST$ preorders.

Lemma 81. *For all finite LTSs $\mathcal{M}$, $\mathcal{M}'$ over $L = L_I \times L_U$ the following holds: $\mathcal{M} \sqsubseteq^{EX}_{MUST} \mathcal{M}'$ implies $\mathsf{lan}\ \mathcal{M}' \subseteq \mathsf{lan}\ \mathcal{M}$.*

Proof. If $\gamma = \epsilon$, then trivially $\gamma \in \mathsf{lan}\ \mathcal{M}$. Suppose $\gamma = \gamma' x \in \mathsf{lan}\ \mathcal{M}' \setminus \mathsf{lan}\ \mathcal{M}$; then we can produce the following derivation:

$\gamma' x \notin \mathsf{lan}\ \mathcal{M}$

$\Rightarrow$ {Def. of experimenter $Ex(\gamma', x)$}

$\perp \notin \text{Result}(Ex(\gamma', x), \mathcal{M})$

$\Rightarrow$ $\{\mathcal{M} \sqsubseteq^{EX}_{MUST} \mathcal{M}'\}$

$\perp \notin \text{Result}(Ex(\gamma', x), \mathcal{M}')$

$\Rightarrow$ {Def. of experimenter $Ex(\gamma', x)$}

$\gamma' x \notin \mathsf{lan}\ \mathcal{M}'$

which is a contradiction, since we assumed $\gamma \in \mathsf{lan}\ \mathcal{M}'$. □

Lemma 82. *For all finite LTSs $\mathcal{M}$, $\mathcal{M}'$ over $L = L_I \times L_U$ if $\mathcal{M} \sqsubseteq^{EX}_{MUST} \mathcal{M}'$ then $\mathcal{M} <<_{MUST} \mathcal{M}'$.*

Proof. Let us assume $\mathcal{M} = (S, s_{\text{in}}, L, \rightarrow)$ and $\mathcal{M} = (S', s'_{\text{in}}, L, \rightarrow)$ for $L = L_I \times L_U$. We must show that $mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma)) \subset\subset mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$. Note that we only need to consider sequences γ which are in $\mathsf{lan}\ \mathcal{M}'$, because if $\gamma \notin \mathsf{lan}\ \mathcal{M}'$ then $\mathsf{AS}\ \mathcal{M}'\ \gamma = \emptyset$, and thus $mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma)) = \emptyset$, and then the relation $\subset\subset$ trivially holds. So, consider $\gamma \in \mathsf{lan}\ \mathcal{M}'$. From Lemma 81 we know that $\gamma \in \mathsf{lan}\ \mathcal{M}$ and thus $\mathsf{AS}\ \mathcal{M}\ \gamma \neq \emptyset$ and also $mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)) \neq \emptyset$. Now we can continue the proof by deriving a contradiction if we assume that $mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma)) \subset\subset mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$ does *not* hold. Under this assumption, by the definition of $\subset\subset$ and considering that both $mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$ and $mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma))$ are non-empty, we can assume that there exists a set $R \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma))$ such that $Z \not\subseteq R$ for all $Z \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$. Now we first show that in each set $Z \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$ we can choose an element (i, u) in such a way that it is not only different from all elements in the set R, but also such that the input part i is different from all input parts of elements in R. We show this by contradiction: for any $Z \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$we assume that it is impossible to choose such an element and we reach a contradiction. For ease of notation, let $mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ s))$ be the set $\{Z_1, \ldots, Z_k\}$. So, now suppose that Z_i differs from R only by elements that differ only in their output part, that is $IN\ Z_i \ \subseteq\ IN\ R \ \wedge\ \exists (i, u) \in Z_i.\ (i, u) \notin R$. Because of the definitions of mfs and $\mathbf{c}$, for each Z_i one of the following cases applies:

(i) $Z_i \in \mathsf{AS}\ \mathcal{M}\ \gamma$;
(ii) $Z_i \in \mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma) \setminus \mathsf{AS}\ \mathcal{M}\ \gamma$;
(iii) $Z_i \in mfs\ (\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)) \setminus \mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)$.

Case (i). *Suppose* $Z_i \in \mathsf{AS}\ \mathcal{M}\ \gamma$.

We first show that $\bigcup_{K' \in \mathsf{AS}\ \mathcal{M}'\ \gamma} K' \subseteq \bigcup_{K \in \mathsf{AS}\ \mathcal{M}\ \gamma} K$. For $\mathsf{AS}\ \mathcal{M}'\ \gamma = \emptyset$ or if $\mathsf{AS}\ \mathcal{M}'\ \gamma = \{\emptyset\}$ this is trivial. For $\mathsf{AS}\ \mathcal{M}'\ \gamma$ containing a non-empty set we can derive:

$$
\begin{array}{ll}
 & x \in \bigcup_{K' \in \mathsf{AS}\ \mathcal{M}'\ \gamma} K' \\
\Rightarrow & \{\text{Set theory}\} \\
 & \exists K' \in \mathsf{AS}\ \mathcal{M}'\ \gamma.\ x \in K' \\
\Rightarrow & \{\text{Def. of } \mathsf{lan} \text{ and of } \mathsf{AS}\} \\
 & \gamma x \in \mathsf{lan}\ \mathcal{M}' \\
\Rightarrow & \{\mathcal{M} \sqsubseteq^{EX}_{MUST} \mathcal{M}', \text{Lemma } 81\} \\
 & \gamma x \in \mathsf{lan}\ \mathcal{M} \\
\Rightarrow & \{\text{Def. of } \mathsf{AS}\} \\
 & \exists K \in \mathsf{AS}\ \mathcal{M}\ \gamma.\ x \in K \\
\Rightarrow & \{\text{Set theory}\} \\
 & x \in \bigcup_{K \in \mathsf{AS}\ \mathcal{M}\ \gamma} K
\end{array}
$$

Now we can show that we can find in $\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)$ a set T that extends Z_i with exactly those elements in R which have the same input part as those in Z_i. We can do this because T is an intermediate set between Z_i—which is in $\mathsf{AS}\ \mathcal{M}\ \gamma$—and $\bigcup_{K \in \mathsf{AS}\ \mathcal{M}\ \gamma} K$—which is an element of $\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)$—and which contains all elements that are in R since we showed $\bigcup_{K' \in \mathsf{AS}\ \mathcal{M}'\ \gamma} K' \subseteq \bigcup_{K \in \mathsf{AS}\ \mathcal{M}\ \gamma} K$. Formally, $T = Z_i \cup \{(i, u) \mid i \in INZ_i \ \wedge\ (i, u) \in R\}$.

It is now easy to see that the set $Z = \{(i, u) \mid i \in INZ_i \ \wedge\ (i, u) \in R\}$ is an element of $mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$. In fact $Z \subseteq T \in \mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)$; moreover Z is functional, since $Z \subseteq R$, and it is maximal. This last fact can be proved by contradiction: suppose there exists an element $(i, u) \in T \setminus Z$; then by definition of Z, since $IN\ T = IN\ Z$, there exists u' such that $(i, u') \in Z$ and since Z is functional we get $u = u'$. So, in the end we found $Z \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$ with $Z \subseteq R$. This contradicts our original assumption that there exists $R \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma))$ such that $Z \not\subseteq R$ for all $Z \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$.

Case (ii). *Suppose* $Z_i \in \mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma) \setminus \mathsf{AS}\ \mathcal{M}\ \gamma$ *and such that* $IN\ Z_i \subseteq IN\ R$.

In this case the reasoning is the same as in case (i).

Case (iii). *Suppose* $Z_i \in mfs\ (\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)) \setminus \mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)$.

In that case, by definition of mfs, we know that there exists a set $K \in \mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)$ such that $IN\ Z_i = IN\ K$ and then $IN\ K \subseteq IN\ R$. For K we can setup a reasoning like in case (ii) leading to the fact that there will exist a set in $mfs\ (\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$ which is a subset of R which is in contradiction with the assumptions, and thus such an Z_i cannot exist.

This ends the proof for each of the cases for Z_i and shows that in all sets $Z \in mfs\ (\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$ we can choose an element, with an input part different from all those of elements of R. And because we can find such an element in every such Z we can also find it in each set $A \in \mathsf{AS}\ \mathcal{M}\ \gamma$. In fact, let A' be a set in $\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma)$. If it is functional then $A' = Z$ for some Z as above. If it is not functional then it includes some functional set Z as above. In particular this holds for all $A \in \mathsf{AS}\ \mathcal{M}\ \gamma$.

Now let us choose in each $Z_i \in mfs\ (\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$ one element x_i such that the input part of x_i is not in INR. Let us call X the set of input parts of all this elements x_i, that is $X = \{i \mid (i, u) \in \{x_1, \ldots, x_k\}\} = \{i_1, \ldots, i_k\}$. Clearly, $\perp \notin \mathrm{Result}(Ex(\gamma, X), \mathcal{M})$ because for each $A \in \mathsf{AS}\ \mathcal{M}\ \gamma$ there exists an element (i, u) with $i \in X$ as shown before. On the other hand, as we will show below, $\perp \in \mathrm{Result}(Ex(\gamma, X), \mathcal{M}')$ which contradicts the hypothesis $\mathcal{M} \sqsubseteq^{EX}_{MUST} \mathcal{M}'$.

What we have to show is that there exists a state s' of $\mathcal{M}'$ which can be reached by s'_{in} by performing γ such that the following computation, where $Ex(X)$ is shown in Fig. B.8, is unsuccessful:

$$Ex(\gamma, X)\ ||\ s'_{\mathrm{in}} \rightsquigarrow \ldots \rightsquigarrow Ex(X)\ ||\ s'$$

Recall that $\gamma \in \mathsf{lan}\,\mathcal{M}'$. So, the fact that the above computation is unsuccessful means that there exists s' such that $IN(\mathsf{S}\ s'\ \epsilon) \cap X = \emptyset$.

There are three possibilities:

(i) $R \in \mathsf{AS}\ \mathcal{M}'\ \gamma$ which means that there exists an s' such that $(\mathsf{S}\ s'\ \epsilon) = R \in \mathsf{AS}\ \mathcal{M}'\ \gamma$. And we know that $(IN\ R) \cap X = \emptyset$.

(ii) $R \in \mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma) \setminus \mathsf{AS}\ \mathcal{M}'\ \gamma$. By definition of closure, this implies that there exists an s' such that $(\mathsf{S}\ s'\ \epsilon) \in \mathsf{AS}\ \mathcal{M}'\ \gamma$ and $(\mathsf{S}\ s'\ \epsilon) \subseteq R$. So, $IN(\mathsf{S}\ s'\ \epsilon) \subseteq IN\ R$, but again we know that $(IN\ R) \cap X = \emptyset$.

(iii) $R \in mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma)) \setminus (\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma) \cup \mathsf{AS}\ \mathcal{M}'\ \gamma)$. This means that there exists a set $K \in \mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma)$ such that $R \in mfs\{K\}$. By definition of closure, this implies that there exists an s' such that $(\mathsf{S}\ s'\ \epsilon) \in \mathsf{AS}\ \mathcal{M}'\ \gamma$ and $(\mathsf{S}\ s'\epsilon) \subseteq K$. So, $IN(\mathsf{S}\ s'\ \epsilon) \subseteq IN\ K$. Since R is a maximal functional subset of K we know that $IN\ K = IN\ R$, so again $(\mathsf{S}\ s'\ \epsilon) \subseteq R$ with $IN\ R \cap X = \emptyset$.

This proves the lemma. □

$$\boxed{Ex(\{i_1, \ldots, i_k\}) \triangleq (i_1; \lambda x.\tau; \mathcal{W}) + \ldots + (i_k; \lambda x.\tau; \mathcal{W})}$$

Fig. B.8. Definition of $Ex(X)$.

The following is an obvious corollary of Lemma 81:

Corollary 83. *For all finite LTSs $\mathcal{M}$, $\mathcal{M}'$ over $L = L_I \times L_U$ the following holds*: $\mathcal{M} \sqsubseteq_{MUST} \mathcal{M}'$ *implies* $\mathsf{lan}\ \mathcal{M}' \subseteq \mathsf{lan}\ \mathcal{M}$.

Proof of Proposition 26. We have to prove that $\mathsf{T_{FAA}}\ \mathcal{M}$ is finite and deterministic and that for all $\gamma \in \mathsf{lan}\ (\mathsf{T_{FAA}}\ \mathcal{M})$, the node of $\mathsf{T_{FAA}}\ \mathcal{M}$ identified by γ is labeled by $(mfs\ A)$ for some $(\mathsf{S}\ (\mathsf{T_{FAA}}\ \mathcal{M})\ \gamma)$-set A.

Finiteness of $\mathsf{T_{FAA}}\ \mathcal{M}$ is guaranteed by the fact that $\mathcal{M}$ is finite by hypothesis, by the fact that the "Subset Construction" algorithm for finite automata determinization, applied at step 1 of the algorithm of Fig. 7, returns a finite automaton (see [1]), and by items (a), (b), and (c) of step 2 of the algorithm of Fig. 7. In particular, item (c) also guarantees that $\mathsf{lan}\ (\mathsf{T_{FAA}}\ \mathcal{M}) = \mathsf{lan}\ \mathcal{M} \subseteq L^*$. Moreover, by item (d) of step 2 of the algorithm, for each $\gamma \in L^* \setminus (\mathsf{lan}\ (\mathsf{T_{FAA}}\ \mathcal{M}))$, we have $\mathsf{AS_{FAA}}\ (\mathsf{T_{FAA}}\ \mathcal{M})\ \gamma = \emptyset$, while, for $\gamma \in (\mathsf{lan}\ (\mathsf{T_{FAA}}\ \mathcal{M}))$, we have $\mathsf{AS_{FAA}}\ (\mathsf{T_{FAA}}\ \mathcal{M})\ \gamma = mfs\ (\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$. It is easy to see that $(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$ is an $(\mathsf{S}\ (\mathsf{T_{FAA}}\ \mathcal{M})\ \gamma)$-set. In fact, the "Subset Construction" algorithm guarantees that, for all $\gamma \in \mathsf{lan}\ \mathcal{M}$ $(\mathsf{S}\ (\mathsf{T_{FAA}}\ \mathcal{M})\ \gamma) = (\mathsf{S}\ \mathcal{M}\ \gamma)$; moreover, by definition, $(\mathsf{AS}\ \mathcal{M}\ \gamma)$ is a set of subsets of $(\mathsf{S}\ \mathcal{M}\ \gamma)$; $(\mathsf{AS}\ \mathcal{M}\ \gamma)$ is finite because $\mathcal{M}$ is finite and it is non-empty, by definition, for $\gamma \in (\mathsf{lan}\ \mathcal{M})$. It is finally easy to see that point (i) of Definition 23 is guaranteed by the definition of $\mathbf{c}$ since $(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$ contains only the elements of $(\mathsf{AS}\ \mathcal{M}\ \gamma)$ (point (i) of Definition 17), which are subsets of $(\mathsf{S}\ \mathcal{M}\ \gamma)$ by definition of $(\mathsf{AS}\ \mathcal{M}\ \gamma)$, and their union and sub-/super-set closures (points (ii) and (iii) of Definition 17). Similarly, point (ii) of Definition 23 is satisfied, as it can be seen by observing that each element of $(\mathsf{S}\ \mathcal{M}\ \gamma)$ is also an element of some set in $(\mathsf{AS}\ \mathcal{M}\ \gamma)$ by definition of the above sets. Finally, points (iii) and (iv) of Definition 23 are clearly satisfied due to points (ii) and (iii) of Definition 17. □

Proof of Theorem 27

$\mathsf{T_{FAA}}\ \mathcal{M}\ \le_{FAA}\ \mathsf{T_{FAA}}\mathcal{M}'$

$\Leftrightarrow$ {Def. of $\le_{FAA}$}

$\mathsf{lan}\ (\mathsf{T_{FAA}}\ \mathcal{M}) = \mathsf{lan}\ (\mathsf{T_{FAA}}\ \mathcal{M}') \wedge \forall \gamma \in \mathsf{lan}(\mathsf{T_{FAA}}\ \mathcal{M}).\ \mathsf{AS_{FAA}}\ (\mathsf{T_{FAA}}\ \mathcal{M}')\ \gamma \subseteq \mathsf{AS_{FAA}}\ (\mathsf{T_{FAA}}\ \mathcal{M})\ \gamma$

$\Leftrightarrow$ {Def. of $\mathsf{T_{FAA}}$}

$\mathsf{lan}\ \mathcal{M} = \mathsf{lan}\ \mathcal{M}' \wedge \forall \gamma \in \mathsf{lan}\ \mathcal{M}.\ mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma)) \subseteq mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$

$\Leftrightarrow$ {Lemma 84 below ; Notice that $\mathsf{lan}\ \mathcal{M} = \mathsf{lan}\ \mathcal{M}'$ implies $\bigcup_{x\in(\mathsf{AS}\ \mathcal{M}'\ \gamma)} x = \bigcup_{x\in(\mathsf{AS}\ \mathcal{M}\ \gamma)} x$}

$\mathsf{lan}\ \mathcal{M} = \mathsf{lan}\ \mathcal{M}' \wedge \forall \gamma \in \mathsf{lan}\ \mathcal{M}.\ mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma)) \subset\subset mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$

$\Leftrightarrow$ {$\gamma \notin \mathsf{lan}\ \mathcal{M} = \mathsf{lan}\ \mathcal{M}'$ iff $\mathsf{AS}\ \mathcal{M}\ \gamma = \mathsf{AS}\ \mathcal{M}'\ \gamma = \emptyset$ from the def. of AS}

$\mathsf{lan}\ \mathcal{M} = \mathsf{lan}\ \mathcal{M}' \wedge \forall \gamma \in L^*.\ mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma)) \subset\subset mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$

$\Leftrightarrow$ {Def. of $<<_{MUST}$; Theo. 21 (b) and Corollary 83}

$\mathsf{lan}\ \mathcal{M} \subseteq \mathsf{lan}\ \mathcal{M}' \wedge \forall \gamma \in L^*.\ mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}'\ \gamma)) \subset\subset mfs(\mathbf{c}\ (\mathsf{AS}\ \mathcal{M}\ \gamma))$

$\Leftrightarrow$ $\quad$ {Def. of $<<_{MAY}$, $<<_{MUST}$, $<<$}

$\mathcal{M} << \mathcal{M}'$. $\square$

In the proof above the following lemma have been used.

Lemma 84. *For X, Y finite sets of finite subsets of L such that $\bigcup_{x \in X} x = \bigcup_{y \in Y} y$ the following holds*: $mfs(\mathbf{c}\ X) \subset\subset mfs(\mathbf{c}\ Y)$ *iff* $mfs(\mathbf{c}\ X) \subseteq mfs(\mathbf{c}\ Y)$.

Proof. The proof consists of two parts, one for each implication.
(i) $\Leftarrow$-part: trivial.
(ii) $\Rightarrow$-part uses Lemma 85 below

$x \in mfs(\mathbf{c}\ X)$

$\Rightarrow$ $\quad$ {Def. of $\subset\subset$}

$\exists y \in mfs(\mathbf{c}\ Y).\ y \subseteq x$

$\Rightarrow$ $\quad$ {Def. of mfs}

$\exists z \in \mathbf{c}\ Y.\ y \in mfs\{z\} \wedge y \subseteq x$

$\Rightarrow$ $\quad$ {Lemma 85 below}

$(x \cup z) \in \mathbf{c}\ Y \wedge x \in mfs\{x \cup z\}$

$\Rightarrow$ $\quad$ {Def. of mfs}

$x \in mfs(\mathbf{c}\ Y)$. $\square$

Lemma 85. *For X, Y finite sets of finite sets over L such that $\bigcup_{x \in X} x = \bigcup_{y \in Y} y$ the following holds*: $z \in \mathbf{c}\ Y \wedge y \in mfs\{z\} \wedge y \subseteq x \Rightarrow x \cup z \in \mathbf{c}\ Y \wedge x \in mfs\{x \cup z\}$.

Proof. First we prove that $x \cup z \in \mathbf{c}\ Y$. We know that $z \in \mathbf{c}\ Y$ and obviously $z \subseteq x \cup z$. Moreover, from the definition of closure we know that $(\bigcup_{y \in Y} y) \in \mathbf{c}\ Y$. So:

$x \subseteq \bigcup_{v \in X} v$

$\Rightarrow$ $\quad$ {By hypothesis $\bigcup_{v \in X} v = \bigcup_{w \in Y} w$}

$x \subseteq \bigcup_{w \in Y} w$

$\Rightarrow$ $\quad$ {$z \in \mathbf{c}\ Y$}

$x \subseteq \bigcup_{w \in Y} w \wedge z \in \mathbf{c}\ Y$

$\Rightarrow$ $\quad$ {$z \in \mathbf{c}\ Y \Rightarrow z \subseteq \bigcup_{w \in Y} w$ by def. of closure}

$x \cup z \subseteq \bigcup_{w \in Y} w$

$\Rightarrow$ $\quad$ {$z \subseteq x \cup z$}

$z \subseteq (x \cup z) \wedge (x \cup z) \subseteq \bigcup_{w \in Y} w$

$\Rightarrow$ {Def. of $\mathbf{c}$ }

$x \cup z \in \mathbf{c}\ Y$

Now we prove that $x \in mfs\{x \cup z\}$. The proof is by derivation of a contradiction. For finite set w over L, we let $func\ w$ denote the predicate $\forall (i_1, u_1), (i_2, u_2) \in w.\ i_1 = i_2 \Rightarrow u_1 = u_2$. Suppose $x \notin mfs\{x \cup z\}$:

$x \notin mfs\{x \cup z\}$

$\Rightarrow$ {Def. of mfs; x is functional}

$\exists k \subseteq x \cup z.\ x \subset k \wedge func\ k$

$\Rightarrow$ {Set theory}

$\exists a \in z \setminus x.\ a \in k$

$\Rightarrow$ $\{y \in mfs\{z\} \wedge x \cap z = y$ see note below; Set theory$\}$

$y \subset k \cap z$

$\Rightarrow$ $\{k \cap z$ is functional since k is functional; $k \cap z \subseteq z\}$

$y \notin mfs\{z\}$

The fact that we derive that y is not in $mfs\{z\}$ is in contradiction with the assumptions. So we proved $x \in mfs\{x \cup z\}$. Note that in the one but last step in the proof above we used $y = x \cap z$. The reason is that $y \subseteq x$ and $y \in mfs\{z\}$ by hypothesis and this last fact implies $y \subseteq z$. Thus $y \subseteq x \cap z$. Moreover, x is functional so also $x \cap z$ must be functional and of course $x \cap z \subseteq z$. But $y \in mfs\{z\}$ so it cannot be $y \subset (x \cap z)$. $\square$

B.3. Proofs related to Section 5

Proof of Lemma 37

Part i: Follows directly from Lemma 76 (i).

Part ii: $\mathcal{C} \xrightarrow{e/\Sigma}_{\Sigma} \mathcal{C}'$

$\Rightarrow$ {Second rule of Def. 36}

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

$\Rightarrow$ {Lemma 76 (ii)}

$\mathcal{C} = \mathcal{C}'$

Part iii: By contradiction. Suppose there exist $\mathcal{C}' \in \mathrm{Conf}_H$ and $\mathcal{E} \in \Theta_E$ such that $\mathcal{C} \xrightarrow{e/\mathcal{E}}_{\Sigma} \mathcal{C}'$ while also $\mathcal{C} \xrightarrow{e/\Sigma}_{\Sigma} \mathcal{C}$. By the first rule of Def. 36, $\mathcal{C} \xrightarrow{e/\mathcal{E}}_{\Sigma} \mathcal{C}'$ would imply $H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{\mathcal{L}} (\mathcal{C}', \mathcal{E})$ for some $\mathcal{L} \neq \emptyset$. But from $\mathcal{C} \xrightarrow{e/\Sigma}_{\Sigma} \mathcal{C}$, by the second rule of Def. 36, we would also have $H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{\emptyset} (\mathcal{C}'', \mathcal{E}')$, which, using Lemma 76 (ii) and (iii), would lead to $\nexists \mathcal{L} \neq \emptyset.\ H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{\mathcal{L}}$, which is a contradiction. $\square$

Proof of Proposition 37. We first consider the direct implication.

$$(\mathcal{C}, \mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}', (\mathsf{Join}\ \mathcal{E}''\ \mathcal{E}'))$$

$\Rightarrow$ {Def. of $\xrightarrow{\mathcal{L}}$ (see Def. 3)}

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

There are two cases:
$\mathcal{L} \neq \emptyset$ and $\mathcal{L} = \emptyset$.
Case 1: $\mathcal{L} \neq \emptyset$

$\Rightarrow$ {First rule of Def. 36, Def. of ΣQueue$(\mathcal{F})$}

$$\exists e \in E.\ \mathcal{C} \xrightarrow{e/\mathcal{E}'}{}_{\Sigma} \mathcal{C}' \wedge$$

$$\Sigma\mathrm{Queue}(\mathcal{E}) \xrightarrow{e} \lambda X.(X \neq \Sigma \Rightarrow \Sigma\mathrm{Queue}(\mathsf{Join}\ \mathcal{E}''\ X) + X = \Sigma \Rightarrow \Sigma\mathrm{Queue}(\mathcal{E}''))$$

$\Rightarrow$ {Def. 5, Def. of ΣQueue$(\mathcal{F})$, $\mathcal{E}' \in \Theta_E$}

$$(\Sigma\mathrm{Queue}(\mathcal{E}) \mid\mid \mathcal{C}) \rightsquigarrow (\Sigma\mathrm{Queue}(\mathsf{Join}\ \mathcal{E}''\ \mathcal{E}') \mid\mid \mathcal{C}')$$

Case 2: $\mathcal{L} = \emptyset$

$\Rightarrow$ {Lemma 76 (ii)}

$$\exists e \in E.\ H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{\emptyset} (\mathcal{C}', \mathcal{E}') \wedge (\mathsf{Sel}\ \mathcal{E}\ e\ \mathcal{E}'') \wedge \mathcal{E}' = \langle\rangle$$

$\Rightarrow$ {Second rule of Def. 36, Def. of ΣQueue$(\mathcal{F})$}

$$\exists e \in E.\ \mathcal{C} \xrightarrow{e/\Sigma}{}_{\Sigma} \mathcal{C}' \wedge \mathcal{E}' = \langle\rangle \wedge$$

$$\Sigma\mathrm{Queue}(\mathcal{E}) \xrightarrow{e} \lambda X.(X \neq \Sigma \Rightarrow \Sigma\mathrm{Queue}(\mathsf{Join}\ \mathcal{E}''\ X) + X = \Sigma \Rightarrow \Sigma\mathrm{Queue}(\mathcal{E}''))$$

$\Rightarrow$ {Def. 5, Def. of ΣQueue$(\mathcal{F})$}

$$(\Sigma\mathrm{Queue}(\mathcal{E}) \mid\mid \mathcal{C}) \rightsquigarrow (\Sigma\mathrm{Queue}(\mathcal{E}'') \mid\mid \mathcal{C}') \wedge \mathcal{E}' = \langle\rangle$$

$\Rightarrow$ $\{\mathcal{E}' = \langle\rangle \Rightarrow (\mathsf{Join}\ \mathcal{E}''\ \mathcal{E}') = \mathcal{E}''\}$

$$(\Sigma\mathrm{Queue}(\mathcal{E}) \mid\mid \mathcal{C}) \rightsquigarrow (\Sigma\mathrm{Queue}(\mathsf{Join}\ \mathcal{E}''\ \mathcal{E}') \mid\mid \mathcal{C}')$$

Now we consider the reverse implication.

$$(\Sigma\mathrm{Queue}(\mathcal{E}) \mid\mid \mathcal{C}) \rightsquigarrow (\Sigma\mathrm{Queue}(\mathsf{Join}\ \mathcal{E}''\ \mathcal{E}') \mid\mid \mathcal{C}')$$

$\Rightarrow$ {Def. 5, Def. of ΣQueue$(\mathcal{F})$}

$$\exists e \in E, u \in {}^{\Sigma}\Theta_E.\ \mathcal{C} \xrightarrow{e/u}{}_{\Sigma} \mathcal{C}' \wedge (\mathsf{Sel}\ \mathcal{E}\ e\ \mathcal{E}'') \wedge$$

$$\Sigma\mathrm{Queue}(\mathcal{E}) \xrightarrow{e} \lambda X.X \neq \Sigma \Rightarrow \Sigma\mathrm{Queue}(\mathsf{Join}\ \mathcal{E}''\ X) + X = \Sigma \Rightarrow \Sigma\mathrm{Queue}(\mathcal{E}'')$$

There are two cases:
$X \neq \Sigma$ and $X = \Sigma$.

Case 1: $X \neq \Sigma$

$\Rightarrow$ {First rule of Def. 36}

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

$\Rightarrow$ {Def. of $\xrightarrow{\mathcal{L}}$ (see Def. 3)}

$\exists \mathcal{L}.\ (\mathcal{C}, \mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}', (\mathsf{Join}\ \mathcal{E}''\ \mathcal{E}'))$

Case 2: $X = \Sigma$

$\Rightarrow$ {Second rule of Def. 36}

$\exists e \in E, \mathcal{E}' \in \Theta_E.\ H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{\emptyset} (\mathcal{C}', \mathcal{E}') \wedge (\mathsf{Sel}\ \mathcal{E}\ e\ \mathcal{E}'')$

$\Rightarrow$ {Def. of $\xrightarrow{\mathcal{L}}$ (see Def. 3)}

$\exists \mathcal{L}.\ (\mathcal{C}, \mathcal{E}) \xrightarrow{\mathcal{L}} (\mathcal{C}', (\mathsf{Join}\ \mathcal{E}''\ \mathcal{E}'))$. $\square$

Proof of Lemma 40. Let $\gamma \in (\mathsf{lan}\ \mathcal{M})$ and $i \in L_I$ as in the definition of $\sqsubseteq_{\mathsf{co}}$. The only interesting case is when $\mathsf{OUT}\ \mathcal{M}'\ \gamma\ i \neq \emptyset$, in which case, for all $u \in \mathsf{OUT}\ \mathcal{M}'\ \gamma\ i$, we get the following derivation:

$\gamma \in (\mathsf{lan}\ \mathcal{M})\ \wedge\ i \in L_I\ \wedge\ u \in \mathsf{OUT}\ \mathcal{M}'\ \gamma\ i$

$\Rightarrow$ {Lemma 86}

$\gamma(i, u) \in (\mathsf{lan}\ \mathcal{M}')$

$\Rightarrow$ {$(\mathsf{lan}\ \mathcal{M}') \subseteq (\mathsf{lan}\ \mathcal{M})$ by hypothesis}

$\gamma(i, u) \in (\mathsf{lan}\ \mathcal{M})$

$\Rightarrow$ {Lemma 86}

$u \in \mathsf{OUT}\ \mathcal{M}\ \gamma\ i$. $\square$

Proof of Lemma 41. By contradiction; suppose that there exists $\gamma \in (\mathsf{lan}\ \mathcal{M}') \setminus (\mathsf{lan}\ \mathcal{M})$. Let $\bar{\gamma}$ the longest prefix of γ such that $\bar{\gamma} \in (\mathsf{lan}\ \mathcal{M})$; such a prefix exists since at least $\epsilon \in (\mathsf{lan}\ \mathcal{M})$ by definition. Let us assume $\gamma = \bar{\gamma}(i, u)\gamma'$ for some $i \in L'_I, u \in L'_U, \gamma' \in (L'_I \times L'_U)^*$. We can now derive the following:

$\bar{\gamma}(i, u) \in (\mathsf{lan}\ \mathcal{M}')$

$\Rightarrow$ {Lemma 86}

$u \in \mathsf{OUT}\ \mathcal{M}'\ \bar{\gamma}\ i$

$\Rightarrow$ {$\bar{\gamma} \in (\mathsf{lan}\ \mathcal{M})$ by assumption; $i \in L'_I$ by assumption; $L'_I \subseteq L_I$ and $\mathcal{M}' \sqsubseteq_{\mathsf{co}} \mathcal{M}$ by hypothesis}

$u \in \mathsf{OUT}\ \mathcal{M}\ \bar{\gamma}\ i$

$\Rightarrow$ {Lemma 86}

$\bar{\gamma}(i, u) \in (\mathsf{lan}\ \ \mathcal{M})$

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

The above proofs used the following lemma:

Lemma 86. *For $\mathcal{M}$ finite LTS over $L_I \times L_U$, $i \in L_I$, $u \in L_U$ and $\gamma \in (L_I \times L_U)^*$ the following holds*: $u \in (\mathit{OUT}\,\mathcal{M}\ \gamma\ i)$ iff $\gamma(i, u) \in (\mathit{lan}\,\mathcal{M})$.

Proof

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

$\Leftrightarrow$ {Def. of OUT}

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

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

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

$\Leftrightarrow$ {Def. of lan}

$\gamma(i, u) \in (\mathsf{lan}\ s)$. □

Proof of Proposition 44. We first of all observe that each experimenter $\mathcal{U}$ generated by the algorithm of Fig. 14 has a tree-like structure, where leaf nodes correspond to δ states and intermediate nodes with more than one branch correspond to (input) states generated by option (2) of the algorithm. Since, by definition, the algorithm is forced to terminate after a *finite* number of recursive calls, the number of calls in which option (2) is selected is finite. All this implies that the depth of $\mathcal{U}$, i.e. the max number of transitions from the root $\mathcal{U}$ to any δ-leaf is finite. Consequently, in order for $\mathcal{U}$ to be infinite, there must be some intermediate node corresponding to an input state which is infinitely branching. In the sequel we show that this cannot be the case when the algorithm is applied to $\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$ for HA H. We observe that ${}^{\Sigma}\mathsf{LTS}(H)$ is finite. This can be seen with similar arguments as for proving the finiteness of $\mathsf{LTS}(H)$. In fact, H has a finite number of sequential automata, each of them has a finite number of states and a finite number of transitions. By definition, a configuration of ${}^{\Sigma}\mathsf{LTS}(H)$ coincides with a set of states of the sequential automata of H. Since the total number of such states is finite, ${}^{\Sigma}\mathsf{LTS}(H)$ has a finite number of configurations. Moreover, from the definition of the Stuttering Semantics (see Section 5.1) and from that of the Core Semantics (see Fig. A.1) it follows that each step-transition of ${}^{\Sigma}\mathsf{LTS}(H)$ corresponds to firing a set $\mathcal{L}$ of transitions of the sequential automata of H, and such a set must be a finite set $\{t_1, \ldots, t_n\}$ since the total number of transitions of the sequential automata is finite. Furthermore there is a finite number of possibilities for Join-ing the output $\mathcal{E}_1, \ldots, \mathcal{E}_n$ produced by the transitions in $\mathcal{L}$. In conclusion, ${}^{\Sigma}\mathsf{LTS}(H)$ has a finite set of states and a finite set of step-transitions. This brings to the conclusion that the set of *all* possible output values that ${}^{\Sigma}\mathsf{LTS}(H)$ can produce is finite and clearly also the set of all the output values which can occur in the traces of ${}^{\Sigma}\mathsf{LTS}(H)$, i.e. the set $\{u \mid \exists\gamma, i.\ \gamma(i,u) \in (\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H))\}$, is finite. Each set $\{u_1,\ldots,u_k\}$ generated by step 2 of the algorithm of Fig. 14 at each recursion call where step 2 is executed is a subset of the above set of all the output values of ${}^{\Sigma}\mathsf{LTS}(H)$. Consequently, each such set is finite as well. $\square$

Proof of Theorem 45. Let $\mathcal{M} = {}^{\Sigma}\mathsf{LTS}(H)$ and $L = L_I \times L_U$. We proceed separately in the proof of soundness and exhaustiveness.

Part 1: soundness
Suppose there exists test case $\mathcal{U} \in \mathsf{TD}_{L,L'}\ (\mathsf{lan}\ \mathcal{M})$ and implementation $\mathcal{M}'$ over $L' = L'_I \times L'_U$ such that $\mathcal{M}' \sqsubseteq_{\mathsf{co}} \mathcal{M}$ and $\mathcal{V}\ \mathcal{U}\ \mathcal{M}' = \mathbf{fail}$. Using Lemma 87 with $\mathsf{lan}\ \mathcal{M}$ for $\mathcal{F}$, we know that this would imply that there exists $\gamma \in\ (\mathsf{lan}\ \mathcal{M})$, $i \in L_I$ and $u \in L'_U$ such that $u \in \mathsf{OUT}\ \mathcal{M}'\ \gamma\ i$ and $u \notin \mathsf{OUT}^*\ (\mathsf{lan}\ \mathcal{M})\ \gamma\ i$. But then, by Lemma 91, we get also $u \notin \mathsf{OUT}\ \mathcal{M}\ \gamma\ i$ which contradicts $\mathcal{M}' \sqsubseteq_{\mathsf{co}} \mathcal{M}$.
Part 2: exhaustiveness
Suppose $\mathcal{M}' \not\sqsubseteq_{\mathsf{co}} \mathcal{M}$, for implementation $\mathcal{M}'$ over $L' = L'_I \times L'_U$. This means that there exist $\gamma \in (\mathsf{lan}\ \mathcal{M})$, $i \in L_I$, $u \in L'_U$ such that $u \in \mathsf{OUT}\ \mathcal{M}'\ \gamma\ i \setminus \mathsf{OUT}\ \mathcal{M}\ \gamma\ i$. Moreover, $\mathsf{OUT}^*\ (\mathsf{lan}\ \mathcal{M})\ \gamma\ i \neq \emptyset$ because $\mathcal{M}$ is input enabled and $\mathsf{OUT}^*\ (\mathsf{lan}\ \mathcal{M})\ \gamma\ i = \mathsf{OUT}\ \mathcal{M}\ \gamma\ i$ by Lemma 91. Thus we can apply Lemma 88 with $\mathsf{lan}\ \mathcal{M}$ for $\mathcal{F}$ to get the assert. $\square$

Lemma 87. *Let $L = L_I \times L_U$ and $L' = L'_I \times L'_U$. For all $\mathcal{F} \subseteq L^*$, implementation $\mathcal{M}' = (S', s'_{in}, L', \longrightarrow)$, $\mathcal{U} \in \mathsf{TD}_{L,L'}\ \mathcal{F}$, unsuccessful computation $\eta \in$ Comp$(\mathcal{U}, \mathcal{M}')$ there exist $\gamma \in \mathcal{F}$, $i \in L_I$, $u \in L'_U$ such that η runs over $\gamma(i,u)$ and $u \in (\mathsf{OUT}\, s'_{in}\ \gamma\ i) \setminus (\mathsf{OUT}^*\ \mathcal{F}\ \gamma\ i)$.* $\square$

Proof. By induction on the structure of $\mathcal{U}$:
Base case ($\mathcal{U} = \tau; \mathbf{W}; \delta$):
There is no unsuccessful computation in Comp$(\tau; \mathbf{W}; \delta, \mathcal{M}')$ so the assert is trivially proved.

Induction step: ($\mathcal{U} = \bar{i}; \lambda x \ldots$):
We can assume by the Induction Hypothesis that there exists $\bar{i}$ such that $\bar{i} \in L_I \cap L'_I$, $\mathsf{OUT}^*\ \mathcal{F}\ \epsilon\ \bar{i} = \{u_1 \ldots u_k\} \neq \emptyset$ and $\mathcal{U} = \bar{i}; \bar{\mathcal{I}}$ with

$$
\begin{aligned}
\bar{\mathcal{I}} = \lambda x.\ (&x = u_1 \ \Rightarrow\ \mathcal{U}_1\\
&+\\
&\vdots\\
&+\\
&x = u_k \ \Rightarrow\ \mathcal{U}_k\\
&+\\
&x \notin \{u_1,\ldots,u_k\} \ \Rightarrow\ \delta)
\end{aligned}
$$

where $\mathcal{U}_j \in \mathsf{TD}_{L,L'}\ (\mathcal{F}\ \mathsf{after}^*\ (\bar{i}, u_j))$ for $j = 1,\ldots,k$. So every (unsuccessful) computation $\eta \in$ Comp$(\mathcal{U}, \mathcal{M}')$ must have the form

$$\eta = \mathcal{U} \mid\mid s'_{\text{in}} \rightsquigarrow \bar{\mathcal{U}} \mid\mid s' \rightsquigarrow \ldots$$

where $\bar{\mathcal{I}} \overset{\bar{u}}{\hookrightarrow} \bar{\mathcal{U}}$ for some $\bar{u} \in L'_U$ and s' such that $s'_{\text{in}} \xrightarrow{(\bar{i},\bar{u})} s'$. Notice that such $\bar{u}$ and s' exist since $\mathcal{M}'$ is input enabled over L' and $\bar{i} \in L'_I$. We distinguish two cases:

Case 1: $\bar{u} \notin \mathsf{OUT}^* \ \mathcal{F} \ \epsilon \ \bar{i}$

For every such $\bar{u} \notin \mathsf{OUT}^* \ \mathcal{F} \ \epsilon \ \bar{i}$ and s' such that $s'_{\text{in}} \xrightarrow{(\bar{i},\bar{u})} s'$ there is only the unsuccessful computation

$$\eta_1 = \mathcal{U} \ || \ s'_{\text{in}} \rightsquigarrow (\delta \ || \ s')$$

since in this case $\bar{\mathcal{U}} = \delta$, by definition of $\mathcal{U}$. Thus the assert is proved with $\gamma = \epsilon$, $i = \bar{i}$, $u = \bar{u}$: $\bar{u} \in (\mathsf{OUT} \ \mathcal{M}' \ \epsilon \ \bar{i})$, since $s'_{\text{in}} \xrightarrow{(\bar{i},\bar{u})} s'$, and $\bar{u} \notin \mathsf{OUT}^* \ \mathcal{F} \ \epsilon \ \bar{i}$.

Case 2: $\bar{u} \in \mathsf{OUT}^* \ \mathcal{F} \ \epsilon \ \bar{i}$

For every η as above we know that its continuation

$$\eta_2 = \bar{\mathcal{U}} \ || \ s' \rightsquigarrow \ldots$$

is an unsuccessful computation in $\text{Comp}(\bar{\mathcal{U}}, s')$ where $\bar{\mathcal{U}}$, such that $\bar{\mathcal{I}} \overset{\bar{u}}{\hookrightarrow} \bar{\mathcal{U}}$, is element of $\mathsf{TD}_{L,L'}(\mathcal{F} \ \mathsf{after}^*(\bar{i}, \bar{u}))$ and s' input enabled over L'. Thus, for every $\bar{u}$ and s' as above we can apply the Induction Hypothesis with $\mathcal{F} \ \mathsf{after}^* \ (\bar{i}, \bar{u})$, s', $\bar{\mathcal{U}}$ and find $\bar{\gamma} \in \mathcal{F} \ \mathsf{after}^* \ (\bar{i}, \bar{u})$, $i \in L_I$ and $u \in L'_U$ such that η_2 runs over $\bar{\gamma}(i, u)$ and $u \in \mathsf{OUT} \ s' \ \bar{\gamma} \ i$ but $u \notin \mathsf{OUT}^* \ (\mathcal{F} \ \mathsf{after}^* \ (\bar{i}, \bar{u})) \ \bar{\gamma} \ i$.

We now observe that

- $(\bar{i}, \bar{u})\bar{\gamma} \in \mathcal{F}$, by def. of after^*, since $\bar{\gamma} \in \mathcal{F} \ \mathsf{after}^* \ (\bar{i}, \bar{u})$;
- η runs over $(\bar{i}, \bar{u})\bar{\gamma}(i, u)$, since $\eta = \mathcal{U} \ || \ \mathcal{M}' \rightsquigarrow \eta_2, \mathcal{U} \xrightarrow{\bar{i}} \bar{\mathcal{I}}, s'_{\text{in}} \xrightarrow{(\bar{i},\bar{u})} s', \bar{\mathcal{I}} \overset{\bar{u}}{\hookrightarrow} \bar{\mathcal{U}}$ and η_2 runs over $\bar{\gamma}(i, u)$;
- $u \in \mathsf{OUT} \ \mathcal{M}' \ (\bar{i}, \bar{u})\bar{\gamma} \ i$, by Lemma 89, since $s'_{\text{in}} \xrightarrow{(\bar{i},\bar{u})} s'$ and $u \in \mathsf{OUT} \ s' \ \bar{\gamma} \ i$;
- $u \notin \mathsf{OUT}^* \ \mathcal{F} \ (\bar{i}, \bar{u})\bar{\gamma} \ i$, by Lemma 93, since u is not an element of the set $\mathsf{OUT}^* \ (\mathcal{F} \ \mathsf{after}^* \ (\bar{i}, \bar{u})) \ \bar{\gamma} \ i$,

which proves the assert with $\gamma = (\bar{i}, \bar{u})\bar{\gamma}$ and i and u as above. □

Lemma 88. *Let $L = L_I \times L_U$ and $L' = L'_I \times L'_U$. For all $\mathcal{F} \subseteq L^*$, $i \in L_I$, $\gamma \in \mathcal{F}$ such that $\mathsf{OUT}^* \ \mathcal{F} \ \gamma \ i \neq \emptyset$, implementation $\mathcal{M}' = (S', s'_{in}, L', \longrightarrow)$, and $u \in L'_U$ such that $u \in (\mathsf{OUT} \ \mathcal{M}' \ \gamma \ i) \setminus (\mathsf{OUT}^* \ \mathcal{F} \ \gamma \ i)$, there exists $\mathcal{U} \in \mathsf{TD}_{L,L'} \ \mathcal{F}$ such that $\mathcal{V} \ \mathcal{U} \ \mathcal{M}' =$* **fail**.

Proof. We proceed by induction on γ.

Base case ($\gamma = \epsilon$):

By hypothesis we know that $i \in L_I$ and $\mathsf{OUT}^* \ \mathcal{F} \ \epsilon \ i \neq \emptyset$; moreover $i \in L'_I$ by Lemma 86 since $(\mathsf{OUT} \ \mathcal{M}' \ \gamma \ i) \neq \emptyset$ and $\mathcal{M}'$ is an LTS labeled over L'. Thus the following test case $\mathcal{U}$ belongs to $\mathsf{TD}_{L,L'} \ \mathcal{F}$: $\mathcal{U} = i; \bar{\mathcal{I}}$ with

$$\begin{aligned} \bar{\mathcal{I}} = \lambda x. \ (&x = u_1 \ \Rightarrow \ \mathcal{U}_1 \\ &+ \\ &\vdots \\ &+ \\ &x = u_k \ \Rightarrow \ \mathcal{U}_k \\ &+ \\ &x \notin \{u_1, \ldots, u_k\} \ \Rightarrow \ \delta) \end{aligned}$$

where $\{u_1, \ldots, u_k\} = \mathsf{OUT}^* \; \mathcal{F} \in i$ and $\mathcal{U}_j$ is an element of $\mathsf{TD}_{L,L'}$ $(\mathcal{F}$ after^* $(i, u_j))$ for $j = 1, \ldots, k$. Moreover $\bar{\mathcal{I}} \overset{u}{\hookrightarrow} \delta$, since by hypothesis $u \notin \mathsf{OUT}^* \; \mathcal{F} \in i$, and $s'_{\text{in}} \xrightarrow{(i,u)} s'$ for some s', since $u \in \mathsf{OUT} \; s'_{\text{in}} \in i$. Thus we can build the following unsuccessful computation

$$\mathcal{U} \mid\mid s'_{\text{in}} \rightsquigarrow \delta \mid\mid s'$$

which makes $\mathcal{V} \; \mathcal{U} \; \mathcal{M}' = \textbf{fail}$ hold.

Induction step $(\gamma = (\bar{i}, \bar{u})\bar{\gamma})$:

We know that

- $\bar{i} \in L_I$, since $(\bar{i}, \bar{u})\bar{\gamma} \in \mathcal{F} \subseteq L^*$ by hypothesis;
- $\bar{i} \in L'_I$, by Lemma 90, since $\mathsf{OUT} \; \mathcal{M}' \; (\bar{i}, \bar{u})\bar{\gamma} \; i \neq \emptyset$ by hypothesis;
- $\mathsf{OUT}^* \; \mathcal{F} \in \bar{i} \neq \emptyset$, by Lemma 92, since $\mathsf{OUT}^* \; \mathcal{F} \; (\bar{i}, \bar{u})\bar{\gamma} \; i \neq \emptyset$, by hypothesis.

Thus the following test case $\mathcal{U}$ belongs to $\mathsf{TD}_{L,L'} \; \mathcal{F}$: $\mathcal{U} = \bar{i}; \bar{\mathcal{I}}$ with

$$\begin{aligned}\bar{\mathcal{I}} = \lambda x.\, (&x = u_1 \;\Rightarrow\; \mathcal{U}_1\\ &+\\ &\vdots\\ &+\\ &x = u_k \;\Rightarrow\; \mathcal{U}_k\\ &+\\ &x \notin \{u_1, \ldots, u_k\} \;\Rightarrow\; \delta)\end{aligned}$$

where $\{u_1, \ldots, u_k\} = \mathsf{OUT}^* \; \mathcal{F} \in \bar{i}$ and $\mathcal{U}_j$ is an element of $\mathsf{TD}_{L,L'}$ $(\mathcal{F}$ after^* $(\bar{i}, u_j))$ for $j = 1, \ldots, k$.

We observe now that

- $\bar{u} \in \mathsf{OUT}^* \; \mathcal{F} \in \bar{i}$, by Lemma 92, since $\mathsf{OUT}^* \; \mathcal{F} \; (\bar{i}, \bar{u})\bar{\gamma} \; i \neq \emptyset$ by hypothesis;
- $\bar{\gamma} \in \mathcal{F}$ after^* $(\bar{i}, \bar{u})$, by definition of after^*, since $(\bar{i}, \bar{u})\bar{\gamma} \in \mathcal{F}$ by hypothesis;
- $\mathsf{OUT}^* \; (\mathcal{F}$ after^* $(\bar{i}, \bar{u})) \; \bar{\gamma} \; i \neq \emptyset$, by Lemma 93, since $\mathsf{OUT}^* \; \mathcal{F} \; (\bar{i}, \bar{u})\bar{\gamma} \; i \neq \emptyset$ by hypothesis.

Thus, using the Induction Hypothesis, we know that for every s' such that $s'_{\text{in}} \xrightarrow{(\bar{i},\bar{u})} s'$ and u element of $\mathsf{OUT} \; s' \; \bar{\gamma} \; i \setminus \mathsf{OUT}^* \; (\mathcal{F}$ $\mathsf{after}^*(\bar{i}, \bar{u})) \; \bar{\gamma} \; i$, there exists $\bar{\mathcal{U}}$ element of $\mathsf{TD}_{L,L'}(\mathcal{F}$ after^* $(\bar{i}, \bar{u}))$ such that $\mathcal{V} \; \bar{\mathcal{U}} \; \bar{s}' = \textbf{fail}$, which means that there exists an unsuccessful computation

$$\bar{\mathcal{U}} \mid\mid s' \rightsquigarrow \ldots$$

But this in turn implies that for every such $(\bar{i}, \bar{u})$ and every $u \in \mathsf{OUT} \; \mathcal{M}' \; (\bar{i}, \bar{u})\bar{\gamma} \; i \setminus \mathsf{OUT}^* \; \mathcal{F} \; (\bar{i}, \bar{u})\bar{\gamma} \; i$ there is an unsuccessful computation

$$\mathcal{U} \mid\mid s'_{\text{in}} \rightsquigarrow \bar{\mathcal{U}} \mid\mid s' \rightsquigarrow \ldots$$

in $\mathrm{Comp}(\mathcal{U}, \mathcal{M}')$. This last fact follows from the fact that $\mathsf{OUT} \; \mathcal{M}' \; (\bar{i}, \bar{u})\bar{\gamma} \; i = \bigcup_{s' : s'_{\text{in}} \xrightarrow{(\bar{i},\bar{u})} s'} \mathsf{OUT} \; s' \; \bar{\gamma} \; i$, by Lemma 89, and $\mathsf{OUT}^* \; (\mathcal{F}$ after^* $(\bar{i}, \bar{u})) \; \bar{\gamma} \; i = \mathsf{OUT}^* \; \mathcal{F} \; (\bar{i}, \bar{u})\bar{\gamma} \; i$, by Lemma 93. So, $\mathcal{V} \; \mathcal{U} \; \mathcal{M}' = \textbf{fail}$. $\square$

The following lemmas show general properties of operators used in the above proofs and follow from the relevant definitions. The detailed proofs are provided in [14].

Lemma 89. *For all LTS $(S, s_{in}, L_I \times L_U, \longrightarrow)$, $s \in S$, $\gamma, \gamma' \in (L'_I \times L'_U)^*$, $i \in L'_I$ the following holds*: $\mathit{OUT} \; s \; \gamma\gamma' \; i = \bigcup_{s' : s \xrightarrow{\gamma} s'} \mathit{OUT} \; s' \; \gamma' \; i$.

Lemma 90. *For all LTS* $\mathcal{M} = (S, s_{in}, L_I \times L_U, \longrightarrow)$, $s \in S$, $i, i' \in L'_I$, $u \in L'_U$, $\gamma \in (L'_I \times L'_U)^*$ *the following holds*: $\mathsf{OUT}\, s\ (i, u)\gamma\ i' \neq \emptyset \Rightarrow i \in L_I$.

Lemma 91. *For all LTS* $\mathcal{M} = (S, s_{in}, L_I \times L_U, \longrightarrow)$, $\gamma \in (\mathsf{lan}\,\mathcal{M})$, $i \in L_I$ *the following holds*: $\mathsf{OUT}^*\ (\mathsf{lan}\,\mathcal{M})\ \gamma\ i = \mathsf{OUT}\, s_{in}\ \gamma\ i$.

Lemma 92. *For all* $\mathcal{F} \subseteq (L_I \times L_U)^*$ $i, i' \in L_I$, $u \in L_U$, $\gamma \in (L_I \times L_U)^*$ *the following holds*: $\mathsf{OUT}^*\ \mathcal{F}\ (i, u)\gamma\ i' \neq \emptyset \Rightarrow u \in \mathsf{OUT}^*\ \mathcal{F}\ \epsilon\ i$.

Lemma 93. *For all* $\mathcal{F} \subseteq (L_I \times L_U)^*$, $\gamma, \gamma' \in (L_I \times L_U)^*$, $i \in L_I$ *the following holds*: $\mathsf{OUT}^*\ (\mathcal{F}\ \mathsf{after}^*\ \gamma)\ \gamma'\ i = \mathsf{OUT}^*\ \mathcal{F}\ \gamma\gamma'\ i$.

B.4. Proofs related to Section 6

Proof of Theorem 46

Part i:

$\mathcal{C} \xrightarrow{e/\mathcal{E}} \mathcal{C}'$

$\Leftrightarrow$ {Def. 13}

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

$\Leftrightarrow$ {Rule (1) of Def. 36}

$\mathcal{C} \xrightarrow{e/\mathcal{E}}_{\Sigma} \mathcal{C}'$

Part ii:

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

$\Leftrightarrow$ {Def. 13; Logics}

$\nexists \mathcal{L} \subseteq \mathsf{Tr}\ H, \mathcal{C}' \in \mathrm{Conf}_H, \mathcal{E} \in \Theta_E.\ H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{\mathcal{L}} (\mathcal{C}', \mathcal{E})$

$\vee$

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

$\Leftrightarrow$ {Lemma 76 (i) and (ii)}

$H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{\emptyset} (\mathcal{C}, \langle\rangle)$

$\Rightarrow$ {Rule (2) of Def. 36}

$\Leftarrow$ {Rule (2) of Def. 36; Lemma 76 (ii)}

$\mathcal{C} \xrightarrow{e/\Sigma}_{\Sigma} \mathcal{C}$. $\square$

Proof of Lemma 49

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 $\alpha_i \in (E \times \Theta_E)^*$, $\beta_i \in (E \times \{\Sigma\})^*$ 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 37(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 $\gamma \setminus \Sigma = \alpha_0\alpha_1 \ldots \alpha_n \in (E \times \Theta_E)^*$ and using Corollary 47, we get easily $\mathcal{C} \xrightarrow{\gamma\setminus\Sigma} \mathcal{C}'$.

Part ii:

Let $\mathsf{LTS}(H) = (\mathrm{Conf}_H, \mathcal{C}_{\mathrm{in}}, \rightarrow)$ and ${}^{\Sigma}\mathsf{LTS}(H) = (\mathrm{Conf}_H, {}^{\Sigma}\mathcal{C}_{\mathrm{in}}, \rightarrow_{\Sigma})$

$\gamma \in (\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H))$

$\Rightarrow$ {Def. of lan}

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

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

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

$\Rightarrow$ {Def. of lan}

$\gamma \setminus \Sigma \in (\mathsf{lan}\ \mathsf{LTS}(H))$

Part iii:

Directly follows from Corollary 47. □

Proof of Lemma 50. We prove $\mathsf{lan}\ \mathsf{LTS}(H') \subseteq \mathsf{lan}\ \mathsf{LTS}(H)$, that is $\mathsf{LTS}(H') <<_{MAY} \mathsf{LTS}(H)$, which, due to Theorem 21 is equivalent to $\mathsf{LTS}(H') \sqsubseteq\!\!\!\sim_{MAY} \mathsf{LTS}(H)$.

$\gamma \in \mathsf{lan}\ \mathsf{LTS}(H')$

$\Rightarrow$ {Lemma 49 (iii)}

$\gamma \in \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H')$

$\Rightarrow$ {Lemma 41; ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$ and $E' \subseteq E$ by hypothesis}

$\gamma \in \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$

$\Rightarrow$ {Lemma 49 (ii); $\gamma \in \mathsf{lan}\ \mathsf{LTS}(H')$ implies $\gamma \setminus \Sigma = \gamma$}

$\gamma \in \mathsf{lan}\ \mathsf{LTS}(H)$. □

Proof of Lemma 51. We let $\mathsf{LTS}(H) = (\mathrm{Conf}_H, \mathcal{C}_{\mathrm{in}}, \longrightarrow)$, $\mathsf{LTS}(H') = (\mathrm{Conf}_{H'}, \mathcal{C}'_{\mathrm{in}}, \longrightarrow)$, ${}^{\Sigma}\mathsf{LTS}(H) = (\mathrm{Conf}_H, {}^{\Sigma}\mathcal{C}_{\mathrm{in}}, \to_{\Sigma})$, and ${}^{\Sigma}\mathsf{LTS}(H') = (\mathrm{Conf}_{H'}, {}^{\Sigma}\mathcal{C}'_{\mathrm{in}}, \to_{\Sigma})$.

We proceed by contradiction.

Suppose ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$ and that there is an experimenter $\mathcal{U}$ such that $\perp \notin \mathrm{Result}(\mathcal{U}, \mathsf{LTS}(H))$ and $\perp$ belongs to $\mathrm{Result}(\mathcal{U}, \mathsf{LTS}(H'))$. Let

$$\eta = \mathcal{U} \mid\mid \mathcal{C}'_{\mathrm{in}} \rightsquigarrow \ldots$$

be an unsuccessful computation in $\mathrm{Comp}(\mathcal{U}, \mathsf{LTS}(H'))$. We distinguish two cases according to η.

Case 1: η is finite

W.l.g. let us assume

$$\eta = \mathcal{U} \mid\mid \mathcal{C}'_{\mathrm{in}} \rightsquigarrow \ldots \rightsquigarrow \mathcal{U}_n \mid\mid \mathcal{C}'$$

and there exist no $\mathcal{U}_{n+1} \mid\mid \mathcal{C}''$ such that $\mathcal{U}_n \mid\mid \mathcal{C}' \rightsquigarrow \mathcal{U}_{n+1} \mid\mid \mathcal{C}''$. In this case we have a derivation $\mathcal{C}'_{\mathrm{in}} \xrightarrow{\gamma} \mathcal{C}'$ on the side of $\mathsf{LTS}(H')$, with $\gamma = (e_1, \mathcal{E}_1) \ldots (e_k, \mathcal{E}_k) \in (E \times \Theta_E)^*$, and a sequence of output transitions $\mathcal{U}_j \xrightarrow{\mu_j} \mathcal{O}_j$, for $j = 1 \ldots n-1$, such that either $\mu_j = e_i$ and $\mathcal{O}_j \overset{\mathcal{E}_i}{\hookrightarrow} \mathcal{U}_{j+1}$ for some i with $1 \le i \le k$, or $\mu_j = \tau$ and $\mathcal{U}_{j+1} = \mathcal{O}_j$. 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 $\mathcal{U}_n \xrightarrow{\tau}$ since otherwise η could not be a computation, not being maximal. There are two other possibilities left:[10]

Case 1.1: $\forall e \in E.\, \mathcal{U}_n \xrightarrow{e} \Rightarrow \nexists \mathcal{E} \in \Theta_E.\, \mathcal{C}' \xrightarrow{e/\mathcal{E}}$

By Lemma 50[11] we know that $\gamma \in \mathsf{lan}\ \mathsf{LTS}(H)$ since $\gamma \in \mathsf{lan}\ \mathsf{LTS}(H')$, by definition of $\mathsf{lan}\ \mathsf{LTS}(H')$ and ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$ by hypothesis. Moreover, by Lemma 49 (iii), we also know that $\gamma \in \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$. Thus, again by ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$, we get

$$\mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H')\ \gamma\ e \subseteq \mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H)\ \gamma\ e \tag{B.2}$$

since $\gamma \in \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$ and $e \in E$[12].

Moreover, by hypothesis we know that there is no $\mathcal{E}$ such that $\mathcal{C}' \xrightarrow{e/\mathcal{E}}$, so, by Theorem 46, we also know that ${}^{\Sigma}\mathcal{C}' \xrightarrow{e/\Sigma}_{\Sigma}$. Moreover, ${}^{\Sigma}\mathcal{C}'_{\mathrm{in}} \xrightarrow{\gamma}_{\Sigma} {}^{\Sigma}\mathcal{C}'$, by Corollary 47, since $\mathcal{C}'_{\mathrm{in}} \xrightarrow{\gamma} \mathcal{C}'$ and $\gamma \in (E \times \Theta_E)^*$. By definition of OUT, we get $\Sigma \in \mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H')\ \gamma\ e$ so, using relation (B.2) above we can conclude $\Sigma \in \mathsf{OUT}\ {}^{\Sigma}\mathsf{LTS}(H)\ \gamma\ e$. But then, again by definition of OUT, we derive that there exists ${}^{\Sigma}\mathcal{C}$ such that ${}^{\Sigma}\mathcal{C}_{\mathrm{in}} \xrightarrow{\gamma}_{\Sigma} {}^{\Sigma}\mathcal{C}$ and ${}^{\Sigma}\mathcal{C} \xrightarrow{e/\Sigma}_{\Sigma}$, and again by Theorem 46 and its corollary we get $\mathcal{C} \xrightarrow{e/\mathcal{E}}$ for no $\mathcal{E} \in \Theta_E$ and $\mathcal{C}_{\mathrm{in}} \xrightarrow{\gamma} \mathcal{C}$. This means that we can build the following computation

$$\mathcal{U} \mid\mid \mathcal{C}_{\mathrm{in}} \rightsquigarrow \ldots \mathcal{U}_n \mid\mid \mathcal{C}$$

---

[10] 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 $\mathcal{U}_n \xrightarrow{e'}$ for some $e' \in E \setminus E'$ (notice that in this case $\mathcal{U}$ should be over $E \cup E'$) and $\mathcal{U}_n \not\xrightarrow{e}$ for all $e \in E'$. This would mean that η would be maximal but we could not infer ${}^{\Sigma}\mathcal{C}_{\mathrm{in}} \xrightarrow{e'/\Sigma}_{\Sigma}$ and in fact it could very well be ${}^{\Sigma}\mathcal{C}_{\mathrm{in}} \xrightarrow{e'/\mathcal{E}'}_{\Sigma}$ for some $\mathcal{E}' \in \Theta_E$ and this extra step could bring to success so we would not reach contradiction.

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

[12] 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 $E' \subseteq E$.

which is an unsuccessful computation since η above was so. This contradicts $\perp \notin \text{Result}(\mathcal{U}, \mathsf{LTS}(H))$.

Case 1.2: $\mathcal{U}_n \xrightarrow{e}$ for no $e \in E$

By Lemma 50 we know that $\gamma \in \mathsf{lan}\ \mathsf{LTS}(H)$ since $\gamma \in \mathsf{lan}\ \mathsf{LTS}(H')$ and ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$. This means, by definition of $\mathsf{lan}\ \mathsf{LTS}(H)$, that $\mathcal{C}_{\text{in}} \xrightarrow{\gamma} \mathcal{C}$ for some $\mathcal{C}$. But then we can build the following computation:

$$\mathcal{U} \mid\mid \mathcal{C}_{\text{in}} \rightsquigarrow \ldots \mathcal{U}_n \mid\mid \mathcal{C}$$

which is an unsuccessful computation since η above was so. This contradicts $\perp \notin \text{Result}(\mathcal{U}, \mathsf{LTS}(H))$.

Case 2: η is infinite

Also in this case the computation gives rise to one derivation on the side of $\mathsf{LTS}(H')$, $\mathcal{C}'_{\text{in}} \xrightarrow{\gamma}$ and to a sequence of output transitions $\mathcal{U}_j \xrightarrow{\mu_j} \mathcal{O}_j$ on the side of $\mathcal{U}$, which involves infinite string $\gamma' \in (E \times \Theta_E)^{\infty}$, which is equal to γ up to τ moves. We distinguish two cases:

Case 2.1: $\forall n \geq 0.\ \exists m \geq n.$ the m-th element of γ' is not τ

From the operational semantics rules of experimental systems (Definition 5) we get that also γ is infinite. For each finite prefix $\bar{\gamma}$ of γ, by Lemma 50 we get $\bar{\gamma} \in \mathsf{lan}\ \mathsf{LTS}(H)$ since ${}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$ by hypothesis. By definition of $\mathsf{lan}\ \mathsf{LTS}(H)$ this means $\mathcal{C}_{\text{in}} \xrightarrow{\bar{\gamma}}$. Thus we can build infinite computation

$$\mathcal{U} \mid\mid \mathcal{C}_{\text{in}} \rightsquigarrow \ldots$$

using, in each step j exactly the same $\mathcal{U}_j$ appearing in η and the same prefix of γ on which η is running 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 $\mathcal{U}_j \xrightarrow{\tau}$ may hold. In conclusion, also in this case we reach a contradiction since the computation we can build involves the same $\mathcal{U}_j$, for $j \geq 0$, occurring in η, which is unsuccessful.

Case 2.2: $\exists n \geq 0.\ \forall m \geq n.$ the m-th element of γ' is τ

In this case γ is finite and by Lemma 50 we get $\mathcal{C}_{\text{in}} \xrightarrow{\gamma}$. Therefore we can build an unsuccessful computation

$$\mathcal{U} \mid\mid \mathcal{C}_{\text{in}} \rightsquigarrow \ldots$$

as in Case 2.1 reaching a contradiction. $\square$

Proof of Lemma 61

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

$\Leftrightarrow$ {Theorem 21}

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

$\Leftrightarrow$ {Def. 19}

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H) \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H')$$

$\Rightarrow$ {Lemma 40}

$\Leftarrow$ {Lemma 41; $E \subseteq E'$}

$${}^{\Sigma}\mathsf{LTS}(H) \sqsubseteq_{\mathsf{co}} \mathsf{LTS}(H'). \quad \square$$

Proof of Lemma 62

$$^{\Sigma}\mathsf{LTS}(H) \sqsubseteq\!\!\!\sim_{MUST} {}^{\Sigma}\mathsf{LTS}(H')$$

$\Rightarrow$ {Corollary 83}

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$$

$\Rightarrow$ {Lemma 40}

$$^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H). \quad \square$$

Proof of Theorem 65

Part i:

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

$\Leftrightarrow$ {Theorem 21(a)}

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H'') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \wedge {}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$$

$\Rightarrow$ {Lemma 41; $E' \subseteq E$}

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H'') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \wedge \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$$

$\Rightarrow$ {Set Theory}

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H'') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$$

$\Rightarrow$ {Lemma 40}

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

Part ii:

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

$\Rightarrow$ {Corollary 83}

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H'') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \wedge {}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H)$$

$\Rightarrow$ {Lemma 41; $E' \subseteq E$}

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H'') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \wedge \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$$

$\Rightarrow$ {Set Theory}

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H'') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$$

$\Rightarrow$ {Lemma 40}

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

Part iii:

Directly follows from Part (ii) and the fact that $\sqsubseteq\!\!\!\sim$ is stronger than $\sqsubseteq\!\!\!\sim_{MUST}$ by Definition 15.

Part iv:

$$^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H) \wedge {}^{\Sigma}\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MAY} {}^{\Sigma}\mathsf{LTS}(H'')$$

$\Rightarrow$ $\{$Lemma 41, $E' \subseteq E\}$

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H) \wedge {}^{\Sigma}\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MAY} {}^{\Sigma}\mathsf{LTS}(H'')$$

$\Rightarrow$ $\{$Theorem 21(a)$\}$

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H) \wedge \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H) \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H'')$$

$\Rightarrow$ $\{$Set Theory$\}$

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H'')$$

$\Rightarrow$ $\{$Lemma 40$\}$

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

Part v: $\quad {}^{\Sigma}\mathsf{LTS}(H') \sqsubseteq_{\mathsf{co}} {}^{\Sigma}\mathsf{LTS}(H) \wedge {}^{\Sigma}\mathsf{LTS}(H'') \underset{\sim}{\sqsubset}_{MUST} {}^{\Sigma}\mathsf{LTS}(H)$

$\Rightarrow$ $\{$Lemma 41; $E' \subseteq E\}$

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H) \wedge {}^{\Sigma}\mathsf{LTS}(H'') \underset{\sim}{\sqsubset}_{MUST} {}^{\Sigma}\mathsf{LTS}(H)$$

$\Rightarrow$ $\{$Corollary 83$\}$

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H) \wedge \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H) \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H'')$$

$\Rightarrow$ $\{$Set Theory$\}$

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H'')$$

$\Rightarrow$ $\{$Lemma 40$\}$

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

Part vi:
Directly follows from Part (v) and the fact that $\underset{\sim}{\sqsubset}$ is stronger than $\underset{\sim}{\sqsubset}_{MUST}$ by Definition 15. □

Proof of Proposition 67

$$^{\Sigma}\mathsf{LTS}(H) \underset{\sim}{\sqsubset}_{MAY} {}^{\Sigma}\mathsf{LTS}(H')$$

$\Leftrightarrow$ $\{$Theorem 21$\}$

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

$\Leftrightarrow$ $\{$Def. 19$\}$

$$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H) \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H')$$

$\Rightarrow$ $\{\mathsf{lan}\ \mathsf{LTS}(H) \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$ by Lemma 49(iii)$\}$

$$\mathsf{lan}\ \mathsf{LTS}(H) \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H')$$

$\Rightarrow$ $\{(\mathsf{lan}\ \mathsf{LTS}(H)) \setminus \Sigma = \mathsf{lan}\ \mathsf{LTS}(H)$; Lemma 49(ii)$\}$

$\mathsf{lan}\ \mathsf{LTS}(H) \subseteq \mathsf{lan}\ \mathsf{LTS}(H')$

$\Leftrightarrow$ {Def. 19}

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

$\Leftrightarrow$ {Theorem 21}

$\mathsf{LTS}(H) \sqsubseteq_{MAY} \mathsf{LTS}(H')$. □

Proof of Proposition 68

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

$\Rightarrow$ {Corollary 83}

$\mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H') \subseteq \mathsf{lan}\ {}^{\Sigma}\mathsf{LTS}(H)$

$\Rightarrow$ {Lemma 40}

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

$\Rightarrow$ {Lemma 51[13]}

$\mathsf{LTS}(H) \sqsubseteq_{MUST} \mathsf{LTS}(H')$. □

Proof of Proposition 69. The proposition directly follows from Propositions 67 and 68. □

References

[1] A. Aho, J. Ullman, Principles of Compiler Design, Addison-Wesley, 1979.
[2] J. Alilovic-Curgus, S. Vuong, A metric based theory of test selection and coverage, in: A. Danthine, G. Leduc, P. Wolper (Eds.), Protocol Specification, Testing, and Verification, XII, IFIP WG 6.1, North-Holland Publishing Company, 1993, pp. 289–304.
[3] K. Bogdanov, M. Holcombe, H. Singh, Automated test set generation for Statecharts, in: D. Hutter, W. Stephan, P. Traverso, M. Ullmann (Eds.), Applied Formal Methods, LNCS, vol. 1641, Springer-Verlag, 1998, pp. 107–121.
[4] T. Bolognesi, M. Caneve, Squiggles—a tool for the analysis of LOTOS specifications, in: K. Turner (Ed.), Forte '88, North-Holland Publishing Company, 1989, pp. 201–216.
[5] B. Bosik, M. Ümit Uyar, Finite state machines based formal methods in protocol conformance testing: from theory to implementation, Computer Networks and ISDN Systems, North-Holland, 1991, pp. 7–33.
[6] H. Bowman, J. Derrick, A junction between state based and behavioural based specifications, in: P. Ciancarini, A. Fantechi, R. Gorrieri (Eds.), IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Oriented Distributed Systems, Kluwer Academic Publishers, 1999, pp. 213–239., ISBN 0-7923-8429-6.
[7] E. Brinksma, J. Tretmans, L. Verhaard, A framework for test selection, in: B. Jonsson, J. Parrow, B. Pehrson (Eds.), Protocol Specification, Testing, and Verification, XI, IFIP WG 6.1, vol. 22, North-Holland Publishing Company, 1991, pp. 289–304.

[13] The use of Lemma 51 requires $E = E'$.

[8] A. Cavarra, C. Crichton, J. Davies, A. Hartman, T. Jeron, L. Mounier, Using UML automatic test generation, in: J.P. Katoen, P. Stevens (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol. 2280, Springer-Verlag, 2002.
[9] R. De Nicola, Extensional equivalences for transition systems, Acta Inform. (24) (1987) 211–237.
[10] A. Fantechi, S. Gnesi, A. Maggiore, Enhancing test coverage by back-tracing model-checker counterexamples, in: M. Pezzé (Ed.), Proceedings of TACoS 2004, Electronic Notes in Theoretical Computer Science, vol. 116, Elsevier Science Publishers B.V., 2004, pp. 199–211.
[11] E. Farchi, A. Hartman, S. Pinter, Using a model-based test generator to test for standard conformance, IBM Syst. J. 41 (1) (2002) 89–109.
[12] L. Feijs, N. Goga, S. Mauw, J. Tretmans, Test selection, trace distance and heuristics, in: IFIP 14th International Conference on Testing of Communicating Systems, 2002.
[13] J. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, M. Sighireanu, CADP: A protocol validation and verification toolbox, Conference on Computer-Aided Verification, LNCS, vol. 1102, Springer-Verlag, 1996, pp. 436–440.
[14] 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).
[15] S. Gnesi, D. Latella, M. Massink, Modular semantics for a UML Statechart Diagrams kernel and its extension to Multicharts and Branching Time Model Checking, J. Logic Algebr. Program. 51 (1) (2002) 43–75.
[16] S. Gnesi, D. Latella, M. Massink, Formal Test-case Generation for UML Statecharts, in: P. Bellini, S. Bohner, B. Steffen (Eds.), 9th IEEE International Conference on Engineering of Complex Computer Systems, IEEE, IEEE Computer Society Press, 2004, pp. 75–84., ISBN 0-7695-2109-6.
[17] M. Hennessy, Algebraic Theory of Processes, MIT Press, 1988.
[18] R. Hierons, Adaptive testing of a deterministic implemenatation against a nondeterministic finite state machine, Comput. J. 41 (5) (1998) 349–355.
[19] C.A.R. Hoare, Communicating Sequential Processes, Series in Computer Science, Prentice-Hall, 1989.
[20] Intecs and CNR-CNUCE, PRIDE Definition of Changes in UML Notation, Technical Report PRIDE Deliverable 1.2, PRIDE, 02.
[21] Y. Kim, H. Hong, D. Bae, S. Cha, Test cases generation from UML state diagrams, IEE Proc. Softw. IEE 146 (4) (1999) 187–192.
[22] R. Langerak, Transformations and semantics for LOTOS, Ph.D. thesis, University of Twente, 1992.
[23] D. Latella, I. Majzik, M. Massink, Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker, Formal Aspects of Computing. Int. J. Formal Methods 11 (6) (1999) 637–664.
[24] D. Latella, I. Majzik, M. Massink, Towards a formal operational semantics of UML statechart diagrams, in: P. Ciancarini, A. Fantechi, R. Gorrieri (Eds.), IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Oriented Distributed Systems, Kluwer Academic Publishers, 1999, pp. 331–347, ISBN 0-7923-8429-6.
[25] D. Latella, M. Massink, A formal testing framework for UML Statechart Diagrams behaviours: From theory to automatic verification, in: A. Jacobs (Ed.), Sixth IEEE International High-Assurance Systems Engineering Symposium, IEEE Computer Society Press, 2001, pp. 11–22., ISBN 0-7695-1275-5.
[26] D. Latella, M. Massink, On testing and conformance relations of UML Statechart Diagrams Behaviours, in: P.G. Frankl (Ed.), Proceedings of the ACM SIGSOFT 2002 International Symposium on Software Testing and Analysis, Association for Computing Machinery—ACM, 2002, ACM Software Engineering Notes, vol. 27(4), pp. 144–153. ISBN 1-58113-562-9.
[27] D. Latella, M. Massink, H. Baumeister, M. Wirsing, Mobile UML statecharts with localities, in: C. Priami, P. Quaglia (Eds.), LNCS, vol. 3267, Springer-Verlag, 2005, pp. 34–58.
[28] D. Lee, M. Yannakakis, Principles and methods of testing finite state machines—a survey, Proceedings of the IEEE 84 (8) (1996) 1090–1123.
[29] N. Lynch, M. Tuttle, An introduction to input/output automata, CWI Quart. 2 (3) (1989) 219–246.
[30] M. Massink, Functional techniques in concurrency, Ph.D. thesis, University of Nijmegen, February 1996. ISBN 90-9008940-3.
[31] M. Massink, D. Latella, S. Gnesi, Testing UML Statecharts. Technical Report 2005-TR-30, Consiglio Nazionale delle Ricerche, Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo', 2005.
[32] E. Mikk, Y. Lakhnech, M. Siegel, Hierarchical automata as model for statecharts, in: R. Shyamasundar, K. Euda (Eds.), Third Asian Computing Science Conference. Advances in Computing Science—ASIAN'97, LNCS, vol. 1345, Springer-Verlag, 1997, pp. 181–196.
[33] R. Milner, Communication and Concurrency, Series in Computer Science, Prentice-Hall, 1989.

[34] Object Management Group, Inc. OMG Unified Modeling Language Specification—version 1.5, 2003. Available from: <http://www.omg.org/cgi-bin/doc?formal/03-03-01>.

[35] A. Petrenko, N. Yevtushenko, G. von Bochmann, Testing deterministic implementations from nondeterministic FSM specifications, in: B. Baumgarten, H. Burkhardt, A. Giessler (Eds.), Testing of Communication Systems, Chapman & Hall, 1996, pp. 125–140.

[36] S. Pickin, C. Jard, Y. Le Traon, T. Jeron, J. Jezequel, A. Le Guennec, System test synthesis from UML models of distributed software, in: D. Peled, M. Vardi (Eds.), Formal Techniques for Networked and Distributed Systems—FORTE 2002, LNCS, vol. 2529, Springer-Verlag, 2002.

[37] A. Pretschner, O. Slotosh, H. Lötzbeyer, E. Aiglstorfer, Model based testing for real: The Inhouse Card study, in: 6th International ERCIM Workshop on Formal Methods for Industrial Critical Systems, Paris, 2001, pp. 79–94.

[38] T. Chow, Testing software design modeled by finite-state machines, IEEE Trans. Software Eng. SE-4 (3) (1978) 178–187.

[39] The Agedis Project, The Agedis Home Page, 2003. Available from: <http://www.agedis.de/index.shtml>.

[40] J. Tretmans, A Formal Approach to Conformance Testing, Ph.D. thesis, University of Twente, 1992.

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

[42] J. Tretmans, Testing concurrent systems: a formal approach, in: J. Baeten, S. Mauw (Eds.), Concur '99, LNCS, vol. 1664, Springer-Verlag, 1999, pp. 46–65.