Exploring Inconsistencies between Modal Transition Systems

Mathieu Sassolas[1]**, Marsha Chechik**[2]**, Sebastian Uchitel**[3]

[1] Université Pierre & Marie Curie, LIP6/MoVe, CNRS UMR 7606, Paris, France

[2] University of Toronto, Toronto, ON, Canada

[3] U. of Buenos Aires, Argentina and Imperial College, UK

January 18, 2010

Abstract   It is commonplace to have multiple behaviour models that describe the same system but have been produced by different stakeholders or synthesized from different sources. Although in practice such models frequently exhibit inconsistencies, there is a lack of tool support for analyzing them. There are two key difficulties in explaining why two behavioural models are inconsistent: (1) explanations often require branching structures rather than linear traces, or scenarios; and (2) there can be multiple sources of inconsistency and many different ways of explaining each one. In this paper, we present an approach that supports exploration of inconsistencies between Modal Transition Systems, an extension to Labelled Transition Systems. We show how to produce sound graphical explanations for inconsistencies, how to compactly represent all possible explanations in a composition of the models being compared, and how modelers can use this composition to explore the explanations encoded therein.

Keywords: Labelled Transition Systems, inconsistency identification and resolution, μ-calculus, distinguishing property, graphical feedback.

1 Introduction

Modelling system behaviour is a common task in requirements engineering and software systems design. Modeling and analyzing behaviour models helps gain confidence in the understanding of the requirements of the system-to-be and the adequacy of the design with respect to these requirements.

It is commonplace to have multiple behavioural models describing the very same system, but produced by different stakeholders, hence providing different views [NKF94] on the system's behavior. Analysis of the similarities and differences of these views supports behaviour model elaboration: confidence is gained on behaviours that are

common to the multiple views; comprehensiveness is furthered by merging behaviours known to some stakeholders but not to others; common understanding is augmented by analyzing and possibly resolving inconsistencies.

Comparison and composition of behaviour models has been studied extensively. Various notions of equivalence [Hoa85,Mil80] provide a framework for comparison that abstracts away syntactic differences in behaviour descriptions. Refinement notions such as those based on simulation [Mil99] support checking whether one model has been further elaborated than another. Also, merge [UC04] allows combining two partial yet mutually consistent descriptions into a comprehensive description that is a refinement of the models being merged.

Although, notions of equivalence, refinement and (inter-model) consistency are crucial for behaviour model elaboration, and tools that support behaviour modelling (e.g., [DFCU08,CPS93,CCGR99,PB00,Hol97]) little support is provided by existing approaches and tools to understand why two models are mutually inconsistent and hence cannot be merged, or why one model is not a refinement of another.

For instance, model-checkers [DFCU08,CCGR99,Hol97] are capable of performing a variety of automated analyses on behaviour models such as property, refinement, equivalence and consistency checking. Such analyses typically either yield a positive result (the property holds in the model, one model is a refinement of another, a model is equivalent to or consistent with another) or provide feedback in the form of a trace representing a counter-example. While this is an effective feedback for deadlock-freedom and reachability properties, the causes for non-equivalence, non-refinement or inconsistency in behavioural models, especially non-deterministic ones, are defined in terms of simulation relations which are not easily visualized. Such explanations can be given in terms of branching structures [CLJV02], which are hard to understand.

Our aim is to automatically provide graphical feedback explaining causes for non-equivalence, non-refinement or inconsistency. Recognizing that there may be many different explanations for these negative results, we also aim to provide support for the user to select among alternative explanations, choosing the one to explore in more detail.

This paper is set in the context of partial behaviour models, specifically, in that of Modal Transition Systems (MTSs). Partial behaviour models support operational descriptions of system behaviour which distinguish between three types of behaviour: *required*, *proscribed* and *unknown*. This distinction extends the expressiveness of traditional behaviour modelling techniques such as Labelled Transition Systems (LTSs) [Kel76] and StateCharts [Har87], and allows describing, in the same operational model, both an upper and a lower bound to the intended system behaviour. The lower bound represents the behaviour that the system must provide, typically identified through scenario and use-case based description techniques. The upper bound represents the behaviour the system may provide without violating known properties and requirements [UBC09].

The semantics of a partial behaviour model can be thought of as a set of traditional behaviour models. For instance, MTS semantics can be given in terms of sets of LTSs that provide all of the behaviour required by the MTS, do not provide any of the behaviour proscribed by the MTS, and make arbitrary decisions on the MTS's

unknown behaviour. In other words, the semantics is the set of LTSs that are between the upper and the lower bounds on system behaviour described by an MTS.

Intuitively, as more information becomes available, unknown or unclassified behaviour gets changed into either required or proscribed behaviour. The notion of refinement between MTSs captures this intuition formally and provides an elegant way of describing the process of behaviour model elaboration as one in which behaviour information is acquired and introduced into the behaviour model incrementally, gradually refining an MTS until it characterizes a single LTS.

MTSs naturally support conjunction of partial knowledge of system behaviour through the notion of *minimal common refinement* [UC04]: an MTS which composes the information of two mutually *consistent* partial models can be constructed through a *merge* operation that attempts to build "the least refined" common refinement of the models being composed. However, if the MTSs to be merged are mutually inconsistent (there are no LTSs which preserve the required and the proscribed behaviour of both MTSs), it is important for modelers to understand sources of such inconsistencies and eventually to fix them.

In this paper, we present an approach that provides feedback explaining why two MTSs with identical alphabets are mutually inconsistent. The *soundness* of the feedback is based on computing a propositional modal μ-calculus formula which captures a property that distinguishes between the two MTSs: it evaluates to true in one and to false in the other. We then use proofs of why this property holds in one model or fails in the other to provide graphical feedback as branching structures overlaid on top of the original models. Recognizing that multiple explanations for inconsistency can be given, we propose an extension to MTSs which can encode *all* such distinguishing properties, allowing the user to guide the generation of inconsistency feedback. We call this extension *pseudo-merge*.

While our results are presented in the general setting of partial behaviour modeling, they can be applied to explore inconsistencies in traditional modelling formalisms such as LTSs. Specifically, given that bisimulation of LTSs is a special case of refinement of MTSs, the approach can be used to describe causes of non-equivalence of two LTS models.

The rest of this paper is organized as follows: We give an example motivating our work in Section 2. In Section 3, we provide background on LTSs, MTSs, and the merge process in general. In Section 4, we describe how to graphically give the feedback to the user to facilitate comprehension of a human modeller. In Section 5, we show how to produce a pseudo-merge of mutually inconsistent MTSs and use it to compute feedback on the cause of inconsistency in the form of propositional μ-calculus formulas. In Section 6, we present a method for the user to guide the generation of this feedback. We discuss case studies in Section 7. We conclude the paper with a survey of related work (Section 8) as well as conclusions and future work (Section 9).

2 Motivation

To motivate our work, we discuss feedback that can be provided to explain the differences between two behavioural models that describe the process for passing laws in a fictitious assembly. In this section, we omit formalizing the various aspects of our work in order to provide a short and intuitive account of our approach. We do include forward references to where formal definitions of the concepts mentioned in this section are introduced.

Consider state-machine models $\mathcal{A}$ and $\mathcal{B}$ in Figure 1(a)-(b) (for the purpose of this section, ignore the "?" symbols that appear on some labels). They describe a process in which, starting from the initial state 0 or $0'$, texts are produced and, after some debate, either rejected or accepted as *laws* or *acts*. The decision of whether a text should pass as an act or a law depends on complex technical aspects that have been abstracted away using non-determinism (e.g., see transitions from states 0 and $0'$ on *propose*). Since laws and acts are to be applied differently, the protocol for passing them differs as well, which is reflected in the models in Figure 1 in the kinds of texts can be amended and consequently re-proposed.

We now discuss the feedback that would be appropriate to explain the difference between these models and thus might be automatically computed by a tool.

A common approach to providing feedback on behaviour models is to show traces, or executions of the model, that highlight the problem at hand [CGP99]. To show the user a *cause* of inconsistency, one could follow the same approach producing a trace that is possible in one of the models but is forbidden in the other. However, it is not always the case that inconsistent models have different traces. For example, models $\mathcal{A}$ and $\mathcal{B}$ have the same traces, yet they are inconsistent as they disagree on whether amendments can be made on laws-to-be or acts-to-be: In model $\mathcal{A}$, amendments can occur only in state 4, and in this state, the proposal, if accepted, becomes law (state 5). In model $\mathcal{B}$, amendments can only occur in state $4'$ as well, but an accepted proposal becomes an act instead (state $5'$).

Traces such as

$$propose, \ amend, \ propose, \ accept, \ applyLaw, \ \ldots$$

do not reveal the different criteria for applying amendments. Although both models can exhibit the trace, they do so by traversing states (4 and $4'$) that have different *potential* behaviour: from state 4, the behaviour *accept*, *applyLaw* can be exhibited; while from state $4'$, *accept*, *applyAct* can occur.

The natural language explanation given above of why $\mathcal{A}$ and $\mathcal{B}$ are mutually inconsistent refers to the potential behaviour of states 4 and $4'$ which essentially refers to the branching structure of the models. Indeed, as claimed in [CLJV02], a more appropriate form of feedback to use is *branching* structures instead of traces. For instance, consider the structure in Figure 1(c). It conveys that the potential behaviour after *propose* is to *amend*, or to *accept* and *applyLaw*. Such behaviour can be exhibited by model $\mathcal{A}$ – a witness is in Figure 1(d), where transitions providing the behaviour are dashed. However, it cannot be exhibited by model $\mathcal{B}$. The counterexample in Figure 1(e) indicates that the *negation* of this behaviour holds: for every *propose* transition, either *accept*, *applyLaw* or *amend* is not
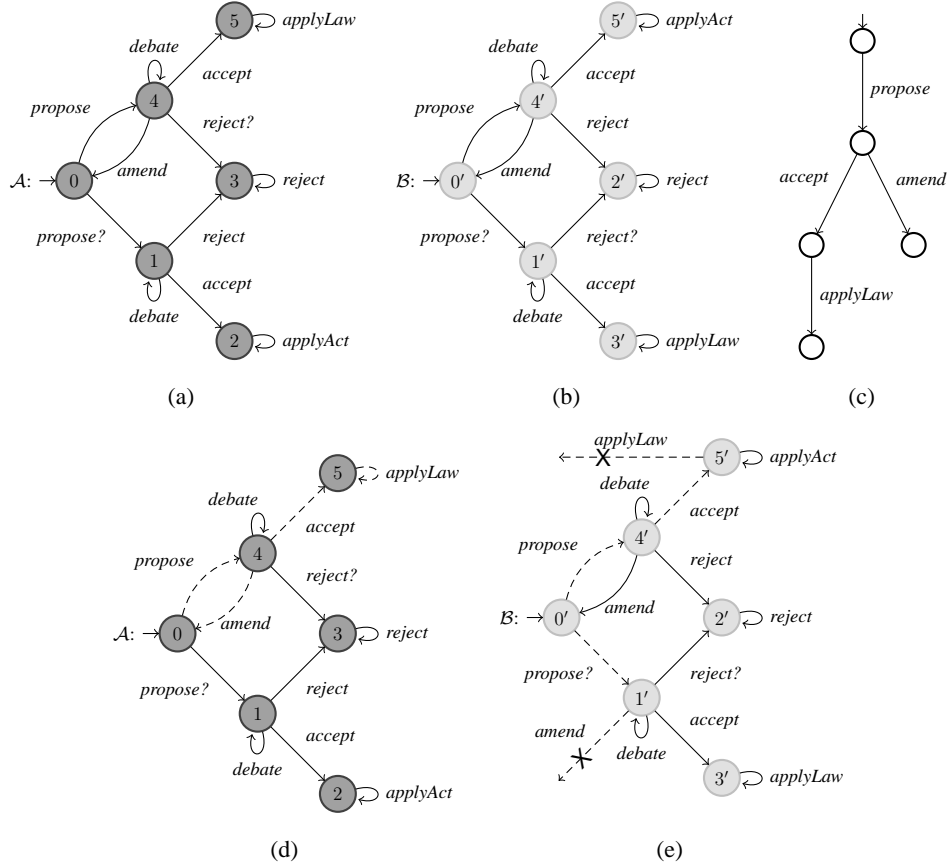
Fig. 1 (a)-(b) Two models of the law-making process; (c) Example showing their inconsistency; (d)-(e) Graphical explanations of inconsistency.

possible. In other words, branching structures can exemplify why two models are inconsistent in general, whereas traces cannot. Furthermore, branching structures can be overlayed over the inconsistent models to better exemplify how one of them can and the other cannot provide the behaviour.

The structure in Figure 1(c) actually corresponds to a *proof* that a specific property holds in the first model but not in the second. The property states that, in the law-making example, there is a *propose* transition leading to a state that allows the *accept*, *applyLaw*, and *amend*; and can be formalized in propositional μ-calculus as

$$\Phi = \langle propose \rangle (\langle accept \rangle \langle applyLaw \rangle \mathbf{t} \ \wedge \ \langle amend \rangle \mathbf{t})$$

In the next sections, we present propositional μ-calculus (see Section 3) and formally define a structure shown in Figure 1(c) as a *proof tree* (see Definition 11 in Section 4).

An important observation is that the formal proof that model $\mathcal{A}$ satisfies property Φ (depicted in Figure 5(a)) can be used to automatically generate the feedback shown in Figures 1(c), 1(d), and 1(e). Figure 1(c) is constructed

by simply ignoring the contents of the states in Figure 5(a)). The other figures can be constructed by overlaying Figure 1(c) onto the original models. Note that proof structures are defined formally in Section 4, and soundness of the graphical feedback produced by our approach is shown in Section 5.

Two models can be inconsistent for several reasons, and even one reason can be explained in a variety of ways. For instance, the disagreement between the law-making models on whether texts with the potential to be passed as laws can be amended, could equally be explained based on the potential to amend acts. Such an explanation also can take the form of a branching structure induced by proof trees of a (different) distinguishing property. Thus, while individual tree structures suffice to explain the difference between two models, there are a number of them that can be proposed, each potentially prompting a different reaction from the modellers. In our trivial example, the first explanation may prompt a discussion on whether laws-to-be can be amended or not, leading to the removal of the transition 4 to 0 or the addition of an *amend* transition from $1'$ to $0'$, while another explanation can prompt a discussion on whether acts-to-be may be amended leading to the removal of a transition from $4'$ to $0'$ or the addition of an *amend* transition from 1 to 0.

Although an explanation of inconsistency can be generated fully automatically, we support user-guided generation of explanations by encoding all sources of inconsistency compactly and allowing exploration of this encoding. Figure 6 depicts a composition of the two law-passing models, $\mathcal{A}$ and $\mathcal{B}$, where states containing $*$ are the disagreement states. This composition, which can be constructed automatically (see Section 5), encodes all of the shortest explanations of why two models are inconsistent. A modeller can generate an explanation of inconsistency by clicking on any transition that leads to a disagreement state (see Section 6). Such explanations are then translated into distinguishing μ-calculus properties, and proofs that they hold in one model and fail in the other get visualized. For example, selecting a transition from $(5, 5')$ to $(5, *)$ generates the feedback we have discussed previously in this section: the distinguishing property Φ is identified and visualized as shown in Figure 1(d)–(e).

3 Background

In this section, we review definitions used in the rest of this paper.

Transition systems. We express models as *Modal Transition Systems* [LT88] which are generalizations of Labelled Transition Systems [Kel76].

Definition 1 (Labelled Transition System) *A* Labelled Transition System *(LTS) is a tuple $\langle S, Act, \Delta, s_0 \rangle$, where S the set of states, Act is the set of actions (or alphabet), $\Delta \subseteq S \times Act \times S$ is the set of transitions, and $s_0 \in S$ is the initial state.*

Definition 2 (Modal Transition System) *A* Modal Transition System *(MTS) is a tuple $\langle S, Act, \Delta^r, \Delta^p, m_0 \rangle$, where S is the set of states, Act is the set of actions (or alphabet), $\Delta^p \subseteq S \times Act \times S$ is the set of possible transitions, $\Delta^r \subseteq \Delta^p$ is the set of required transitions, and $m_0 \in S$ is the initial state.*

We write $m \xrightarrow{\ell}_p m'$ when $(m, \ell, m') \in \Delta^p$ (i.e., there is a possible transition between m and m' on ℓ), $m \xrightarrow{\ell}_r m'$ when $(m, \ell, m') \in \Delta^r$ (required transition), $m \xrightarrow{\ell}_m m'$ when $(m, \ell, m') \in \Delta^p \setminus \Delta^r$ (a transition which is possible but not required, referred to as a *maybe transition*), and $m \xnrightarrow{\ell}$ when $\forall m' \in S, (m, \ell, m') \notin \Delta^p$ (there is no transition on ℓ from m). Note that MTSs where $\Delta^r = \Delta^p$ are LTSs.

For example, Figure 1(a) depicts an MTS. Pictorially, a transition between state 4 and state 3 on *reject?* means that there is a possible transition between these states on action *reject* but there isn't a required transition between these states on this action.

An MTS can be obtained from another one by removing some states and all transitions to and from the removed states, leaving all other transitions untouched. We call the resulting MTS a *subMTS* of the original one:

Definition 3 (SubMTS) *For an MTS $M = \langle S_M, Act, \Delta^r_M, \Delta^p_M, m_0 \rangle$, a subMTS w.r.t. a set of states $S_N \subseteq S_M$ is an MTS $N = \langle S_N, Act, \Delta^r_N, \Delta^p_N, n_0 \rangle$, where $n_0 = m_0$, $\Delta^p_N = \Delta^p_M \cap (S_N \times Act \times S_N)$, and $\Delta^r_N = \Delta^p_N \cap \Delta^r_M$.*

For example, an MTS consisting of states 0 and 4 and transitions on *propose*, *amend* and a self-loop on *debate* is a subMTS of model $\mathcal{A}$ in Figure 1(a).

Propositional μ-calculus. In order to reason over *finite* behaviors of MTSs, we use a propositional subset of the modal 3-valued μ-calculus of [HJS01] that does not include fixpoint operators. We refer to it as $\mathcal{L}^p_\mu$.

Definition 4 (Propositional μ-calculus) *A formula of the* propositional μ-calculus *($\mathcal{L}^p_\mu$) has the grammar*

$$\varphi = \ \mathbf{t} \mid \mathbf{f} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle \ell \rangle \varphi \mid [\ell]\varphi,$$

where ℓ is an action.

$\langle \ell \rangle \varphi$ means that there exists a required transition on ℓ to a state in which φ is satisfied. $[\ell]\varphi$ means that every possible transition on ℓ leads to a state in which φ is satisfied.

For example, a formula

$$\Phi = \langle propose \rangle (\langle accept \rangle \langle applyLaw \rangle \mathbf{t} \ \wedge \ \langle amend \rangle \mathbf{t}),$$

introduced in Section 2, means that after a *propose*, two choices must be available: to do an *accept* followed by *applyLaw* and to do an *amend*.

In the remainder of the paper, when we discuss μ-calculus, we mean the propositional subset defined above. Furthermore, we shall refer to a formula expressed in this logic as a *property*.

$$\llbracket \mathbf{t} \rrbracket_M \triangleq S_M$$

$$\llbracket \mathbf{f} \rrbracket_M \triangleq \emptyset$$

$$\llbracket \varphi \wedge \psi \rrbracket_M \triangleq \llbracket \varphi \rrbracket_M \cap \llbracket \psi \rrbracket_M$$

$$\llbracket \varphi \vee \psi \rrbracket_M \triangleq \llbracket \varphi \rrbracket_M \cup \llbracket \psi \rrbracket_M$$

$$\llbracket \neg \varphi \rrbracket_M \triangleq S_M - \llbracket \varphi \rrbracket_M$$

$$\llbracket \langle \ell \rangle \varphi \rrbracket_M \triangleq \{ s \in S_M \mid \exists s' \in S_M \cdot (s \xrightarrow{\ell}_{\mathsf{r}} s' \wedge s' \in \llbracket \varphi \rrbracket_M) \}$$

$$\llbracket [\ell] \varphi \rrbracket_M \triangleq \{ s \in S_M \mid \forall s' \in S_M \cdot (s \xrightarrow{\ell}_{\mathsf{p}} s' \Rightarrow s' \in \llbracket \varphi \rrbracket_M) \}$$

Fig. 2 2-valued semantics of $\mathcal{L}_\mu^p$ in an MTS $M = \langle S_M, Act, \Delta_M^{\mathsf{r}}, \Delta_M^{\mathsf{p}}, m_0 \rangle$.

Given an MTS $M = \langle S_M, Act, \Delta_M^{\mathsf{r}}, \Delta_M^{\mathsf{p}}, m_0 \rangle$, we write $\llbracket \varphi \rrbracket_M$ to mean a set of states in S_M where formula φ holds; thus, φ holds in state $m \in S_M$ iff $m \in \llbracket \varphi \rrbracket_M$. An often-used notation for this is $M, m \vDash \varphi$. If $m_0 \in \llbracket \varphi \rrbracket_M$, we often write $M \vDash \varphi$. Finally, when a transition system in question is clear from the context, it is often dropped from the notation, so we simply write $\llbracket \varphi \rrbracket$.

The meaning of the other operators is as usual. For example, $\varphi \vee \psi$ holds in state m if either φ or ψ holds in it, or, formally, $m \in \llbracket \varphi \vee \psi \rrbracket$ iff $m \in \llbracket \varphi \rrbracket \cup \llbracket \psi \rrbracket$. That is, the set of states where $\varphi \vee \psi$ holds is the union of those states where φ holds and those where ψ holds. The complete 2-valued semantics of $\mathcal{L}_\mu^p$ is shown in Figure 2. Under this semantics, property Φ holds in model $\mathcal{A}$ and fails in model $\mathcal{B}$ (see Figure 1).

Instead, consider another property

$$\Phi_1 = \langle propose \rangle \langle reject \rangle.$$

While this property clearly holds in model $\mathcal{B}$ (via the path 0', 4', 2'), it does not hold in model $\mathcal{A}$. A *propose* is either followed by a maybe transition on *reject*, or a maybe transition on *propose* is followed by a *reject*. Thus, the desired behaviour is possible in $\mathcal{A}$, just not required. In order to allow us to make this distinction, we use the 3-valued semantics of $\mathcal{L}_\mu^p$, where a property can evaluate to $\mathbf{t}$ (*true*), $\mathbf{f}$ (*false*) and $\perp$ (*maybe* or *unknown*).

Let an MTS $M = \langle S_M, Act, \Delta_M^{\mathsf{r}}, \Delta_M^{\mathsf{p}}, m_0 \rangle$ and an $\mathcal{L}_\mu^p$ property φ be given. Our goal is thus to determine whether definite behaviours of M (i.e., either required or proscribed) are sufficient to ensure that φ holds or fails. In these circumstances, we want φ to evaluate to *true* or *false*, respectively. In all other cases we want φ to evaluate to *maybe*. Let $\llbracket \varphi \rrbracket_M^{\mathbf{t}}$ denote the set of states in M where φ is *true*, and let $\llbracket \varphi \rrbracket_M^{\mathbf{f}}$ denote the set of states where φ is *false*. The set of states where φ is *maybe* is then $S_M \backslash (\llbracket \varphi \rrbracket_M^{\mathbf{t}} \cup \llbracket \varphi \rrbracket_M^{\mathbf{f}})$ (i.e., φ is neither *true* nor *false*).

Definition 5 (3-**valued Semantics of** $\mathcal{L}_\mu^p$) *For an MTS* $M = \langle S_M, Act, \Delta_M^{\mathsf{r}}, \Delta_M^{\mathsf{p}}, m_0 \rangle$ *and a property* φ *in* $\mathcal{L}_\mu^p$, $\llbracket \varphi \rrbracket_M^{\mathbf{t}} \subseteq S_M$ *and* $\llbracket \varphi \rrbracket_M^{\mathbf{f}} \subseteq S_M$ *are defined as shown in Figure 3. We often use abbreviations* $M \vDash \varphi$ *and* $M \vDash \neg \varphi$ *to mean* $m_0 \in \llbracket \varphi \rrbracket_M^{\mathbf{t}}$ *and* $m_0 \in \llbracket \varphi \rrbracket_M^{\mathbf{f}}$, *respectively.*

Note that if $M \vDash \neg \varphi$, we say that φ is *false* in M. This is not the same as $M \nvDash \varphi$, since when φ is not *true* in M, it can evaluate to either *maybe* or *false*.

$$\llbracket \mathbf{t} \rrbracket_M^{\mathbf{t}} \triangleq S_M$$

$$\llbracket \mathbf{t} \rrbracket_M^{\mathbf{f}} \triangleq \emptyset$$

$$\llbracket \bot \rrbracket_M^{\mathbf{t}} \triangleq \emptyset$$

$$\llbracket \varphi \wedge \psi \rrbracket_M^{\mathbf{t}} \triangleq \llbracket \varphi \rrbracket_M^{\mathbf{t}} \cap \llbracket \psi \rrbracket_M^{\mathbf{t}}$$

$$\llbracket \varphi \wedge \psi \rrbracket_M^{\mathbf{f}} \triangleq \llbracket \varphi \rrbracket_M^{\mathbf{f}} \cup \llbracket \psi \rrbracket_M^{\mathbf{f}}$$

$$\llbracket \neg \varphi \rrbracket_M^{\mathbf{t}} \triangleq \llbracket \varphi \rrbracket_M^{\mathbf{f}}$$

$$\llbracket \langle \ell \rangle \varphi \rrbracket_M^{\mathbf{t}} \triangleq \{ s \in S_M \mid \exists s' \in S_M \cdot (s \xrightarrow{\ell}_{\mathsf{r}} s' \wedge s' \in \llbracket \varphi \rrbracket_M^{\mathbf{t}}) \}$$

$$\llbracket \langle \ell \rangle \varphi \rrbracket_M^{\mathbf{f}} \triangleq \{ s \in S_M \mid \forall s' \in S_M \cdot (s \xrightarrow{\ell}_{\mathsf{p}} s' \Rightarrow s' \in \llbracket \varphi \rrbracket_M^{\mathbf{f}}) \}$$

Fig. 3 3-valued semantics of $\mathcal{L}_\mu^p$ in an MTS $M = \langle S_M, Act, \Delta_M^{\mathsf{r}}, \Delta_M^{\mathsf{p}}, m_0 \rangle$.

The semantics of $\mathcal{L}_\mu^p$ defined in Figure 3 is straightforward. For instance, property $\mathbf{t}$ is true in all states (line 1), never *false* (line 2) or *maybe* (line 3); if either property φ and property ψ is *true*, then so is their conjunction (line 4); and if either property φ or property ψ is *false*, then so is their conjunction (line 5). Property $\langle \ell \rangle \varphi$ is *false* if every possible transition leads to a state where φ has value *false* (line 8). This semantics explicitly enumerates states where a formula is *true* and *false*. In all other states, it evaluates to *maybe*.

Under this semantics, property Φ evaluates to *true* in model $\mathcal{A}$ and to *false* in model $\mathcal{B}$, as expected. And property Φ_1 evaluates to *true* in model $\mathcal{B}$ and to *maybe* in model $\mathcal{A}$.

Semantics of the remainder of the operators from Definition 4 is given via negation: $\varphi_1 \vee \varphi_2 = \neg \varphi_1 \wedge \neg \varphi_2$, and $[\ell]\varphi = \neg \langle \ell \rangle \neg \varphi$.

Every formula in $\mathcal{L}_\mu^p$ can be put in a form where negation is applied only to the level of atomic propositions, referred to as *negation normal form*.

Definition 6 (Distinguishing property) *Let MTSs M and N be given. An $\mathcal{L}_\mu^p$ formula φ is a* distinguishing property *iff $M \vDash \varphi$ and $N \vDash \neg \varphi$, or vice versa.*

For example, property Φ is a distinguishing property for models $\mathcal{A}$ and $\mathcal{B}$ whereas Φ_1 is not. Note that if φ is distinguishing, then so is $\neg \varphi$. In the rest of this paper, we usually pick distinguishing properties that evaluate to *true* in the first model and *false* in the second.

Refinement. As mentioned earlier in the paper, we use *refinement* as a relation that captures the notion of one model having more information than another. Intuitively, an MTS N refines an MTS M if N includes all of M's required behavior and does not have any of M's proscribed behavior. That is, only *maybe* behaviour might get changed, either into required or proscribed.

Definition 7 (Refinement) *[LT88] An MTS $N = \langle S_N, Act, \Delta_N^{\mathsf{r}}, \Delta_N^{\mathsf{p}}, n_0 \rangle$ is a* refinement *of an MTS $M = \langle S_M, Act, \Delta_M^{\mathsf{r}}, \Delta_M^{\mathsf{p}}, m_0 \rangle$ over the same alphabet, written $M \preceq N$, iff there exists a refinement relation $\mathcal{R}$ such*
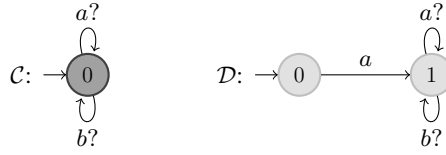
Fig. 4 Two MTSs $\mathcal{C}$ and $\mathcal{D}$ where $\mathcal{D}$ refines $\mathcal{C}$.

that $(m_0, n_0) \in \mathcal{R}$ and $\forall (m, n) \in \mathcal{R}$, the following hold:

$$m \xrightarrow{\ell}_r m' \Rightarrow (\exists n' \cdot n \xrightarrow{\ell}_r n' \wedge (m', n') \in \mathcal{R})$$
$$n \xrightarrow{\ell}_p n' \Rightarrow (\exists m' \cdot m \xrightarrow{\ell}_p m' \wedge (m', n') \in \mathcal{R})$$

The above definition means that N has all required behaviours of M, and N does not introduce any new *maybe* behaviour that was not present in M. The above definition also indicates that refinement is a reflexive and transitive operation.

For example, model $\mathcal{D}$ refines model $\mathcal{C}$ in Figure 4 with $R = \{(0, 0), (0, 1)\}$.

Refinement of a given MTS M can proceed until all behaviours are known, and the resulting LTS is called an *implementation* of M. Since different refinements yield different LTSs, one can view an MTS as the (possibly infinite) set of its implementations denoted $\mathcal{I}(M)$.

Refinement also preserves values of "definite" $\mathcal{L}^p_\mu$ formulas. That means that if a $\mathcal{L}^p_\mu$ formula φ *true* or *false* in an MTS M, it is guaranteed to have the same value in each of its refinements N, as formalized below. However, nothing can be concluded about $\mathcal{L}^p_\mu$ formulas which evaluate to *maybe* in M: they can be *true*, *false* or *maybe* in N.

Property 1 Let MTSs M and N, where $M \preceq N$ be given. Then for each $\mathcal{L}^p_\mu$ property φ, if $M \vDash \varphi$, then $N \vDash \varphi$, and if $M \vDash \neg\varphi$, then $N \vDash \neg\varphi$.

The above property is very important for the use of refinement as the underlying operation supporting development of implementations from partial models: once a value of a property has been established (i.e., it is no longer $\bot$), it will remain so for all implementations of this model.

Consistency. When MTSs represent partial views on the future system, it makes sense to determine whether a pair of views is consistent.

Two MTS models are *consistent* if they allow at least one common implementation. Formally, it means that a common refinement of these two models exists:

Definition 8 (Consistency) *Two MTSs M and N over the same alphabet Act are* consistent *(denoted $\mathrm{Cons}(M, N)$) if there is an MTS P such that $M \preceq P$ and $N \preceq P$.*

Intuitively, MTSs are consistent if they can match each other's behaviour, i.e., there isn't a case where one MTS has a required transition on an action ℓ and the other does not allow a transition on ℓ from this state. Thus, we want to match states of the two MTSs so that when one has a required transition on an action, the other has a possible transition on this action and vice versa. This matching is captured in a *consistency relation*:

Definition 9 (Consistency relation) *[UC04] A* Consistency relation *between two MTSs* $M = \langle S_M, Act, \Delta_M^r, \Delta_M^p, m_0 \rangle$ *and* $N = \langle S_N, Act, \Delta_N^r, \Delta_N^p, n_0 \rangle$ *is a binary relation* $C_{M,N} \subseteq S_M \times S_N$ *such that* $(m_0, n_0) \in C_{M,N}$ *and for all* $\ell \in Act$ *and all* $(m, n) \in C_{M,N}$, *the following hold:*

$$m \xrightarrow{\ell}_r m' \Rightarrow \exists n' \cdot n \xrightarrow{\ell}_p n' \wedge (m', n') \in C_{M,N}$$
$$n \xrightarrow{\ell}_r n' \Rightarrow \exists m' \cdot m \xrightarrow{\ell}_p m' \wedge (m', n') \in C_{M,N}$$

For example, the consistency relation between models $\mathcal{C}$ and $\mathcal{D}$ in Figure 4 is $C_{\mathcal{C},\mathcal{D}} = \{(0, 0), (0, 1)\}$.

Two states are *consistent*, denoted $\mathrm{Cons}(m, n)$, iff they are in a consistency relation. Moreover, consistency of models and presence of a consistency relation are closely related:

Property 2 [FU08] Two MTSs M and N over the same alphabet are consistent if and only if there is a consistency relation between them.

Thus, if we want to prove that two models are consistent, we can do so by identifying a consistency relation between them.

Merge. The process of *merging* two consistent MTSs aims to put together knowledge contained in each of them into a common model.

Definition 10 (Merge) *[UC04] A* merge *of MTSs* M *and* N *with identical alphabets is an MTS* P, *over the same alphabet, such that* P *is a* minimum common refinement *of* M *and* N:

$$(M \preceq P) \wedge (N \preceq P) \wedge (\forall Q \cdot (M \preceq Q \wedge N \preceq Q) \Rightarrow P \preceq Q)$$

Intuitively, any common refinement adds knowledge from N to elaborate *maybe* behaviours of M and vice versa. The fact that the common refinement we are looking for is *minimal* means that no extra knowledge gets introduced. For example, model $\mathcal{C}$ over the alphabet $\{a, b\}$ in Figure 4 has no required or proscribed behaviour – all of its behaviour is *maybe*, and thus merging it with model $\mathcal{D}$ over the same alphabet leaves $\mathcal{D}$ unchanged.

Moreover, by Property 1, $\mathcal{L}_\mu^p$ properties which have value *true* or *false* in the original models have the same value in their merge. For example, property $\langle a \rangle \mathbf{t}$ was *true* in $\mathcal{D}$ and this value is preserved in its merge with $\mathcal{C}$.

Merge and inconsistency. Clearly, inconsistent models cannot be merged as they have no common refinements. Moreover, if models disagree on some property, i.e., it is *true* in one model and *false* in the other, clearly they are inconsistent and cannot be merged.

The theorem below indicates that presence of such distinguishing properties is necessary and sufficient for a pair of models to be inconsistent:

Theorem 1 *MTSs M and N over the same alphabet are inconsistent iff there exists an $\mathcal{L}_\mu^p$ property distinguishing between them.*

This theorem follows from an analogous one in [BCU06] that guarantees the existence of a full μ-calculus property iff the models are inconsistent. The theorem holds for $\mathcal{L}_\mu^p$ because we aim to find a reachable pair of states where models cannot simulate each other's behavior – something that needs only a finite path for MTSs with finite statespaces.

The above theorem allows us to use distinguishing properties for providing feedback for exploring inconsistencies between models – the subject of the rest of this paper.

4 Explaining Inconsistency Graphically

In this section, we describe sound graphical feedback explaining why two models are inconsistent.

As shown in Section 3, inconsistency between two models can be characterized by the existence of a distinguishing $\mathcal{L}_\mu^p$ property. As $\mathcal{L}_\mu^p$ is clearly not an accessible language from a practitioner's perspective, we show how *graphical* feedback, in terms of two directed acyclic graphs (DAG) – each one overlaid on one of the models being compared – can provide an intuitive explanation to inconsistency. These graphs formally correspond to proofs as to why the property does or does not hold in the inconsistent models.

We first define the notion of a proof-tree. For a model M with initial state m_0 and an $\mathcal{L}_\mu^p$ property φ, a *proof-tree* encodes a proof that $M \vDash \varphi$. Each node of the tree consists of a tuple $\langle \text{node, property} \rangle$, with the root labelled with $\langle m_0, \varphi \rangle$, and each transition labelled with an action $\ell \in Act \cup \{\tau\}$. Label τ symbolizes a silent action. A node $\langle m, \varphi \rangle$ has successors $\langle m_i, \varphi_i \rangle$ reached via transitions ℓ_i if $m \vDash \varphi$ follows from $\forall i \cdot m_i \vDash \varphi_i \wedge m \xrightarrow{\ell_i} m_i$. Leaves of a proof-tree can be of the form $\langle m, \mathbf{t} \rangle$ or $\langle m, [\ell]\varphi \rangle$, where there is no transition on ℓ from m.

Definition 11 (Proof-Trees) *Let an MTS $M = \langle S_M, Act, \Delta_M^r, \Delta_M^p, m_0 \rangle$ and an $\mathcal{L}_\mu^p$ property φ such that $M \vDash \varphi$ be given. Further assume that φ is in negation normal form. Then a* proof-tree *for φ in M, denoted Π_M^φ, is a labelled tree $T(m_0, \varphi)$, where T is inductively defined as follows:*
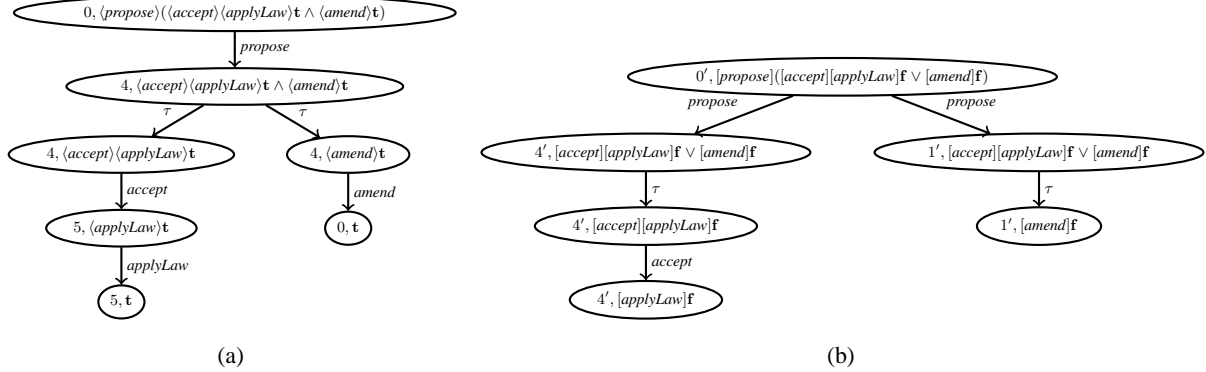
Fig. 5 Proof-trees: (a) for a property Φ in model $\mathcal{A}$; (b) for a property $\neg\Phi$ in model $\mathcal{B}$.

$$
\begin{aligned}
T(m, \mathbf{t}) \quad &= (m, \mathbf{t}) \\
T(m, \langle\ell\rangle\varphi) \quad &= (m, \langle\ell\rangle\varphi) \xrightarrow{\ell} T(m', \varphi), \text{ where } m \xrightarrow{\ell}_r m' \text{ and } m' \vDash \varphi \\
T(m, [\ell]\varphi) \quad &= (m, [\ell]\varphi) \text{ iff } \forall m', (m, \ell, m') \notin \Delta_M^{\mathsf{p}} \text{ or } (m, [\ell]\varphi) \xrightarrow{\ell} \{T(m_1', \varphi), \ldots, T(m_k', \varphi)\}, \\
&\quad \text{where } \{m_i'\}_{1 \le i \le k} \text{ are all states such that } m \xrightarrow{\ell}_{\mathsf{p}} m_i' \text{ and for each } i \in \{1, \ldots, k\}\ m_i' \vDash \varphi \\
T(m, \textstyle\bigwedge_{i=1}^{k} \varphi_i) &= (m, \textstyle\bigwedge_{i=1}^{k} \varphi_i) \xrightarrow{\tau} \{T(m, \varphi_1), \ldots, T(m, \varphi_k)\} \\
T(m, \textstyle\bigvee_{i=1}^{k} \varphi_i) &= (m, \textstyle\bigvee_{i=1}^{k} \varphi_i) \xrightarrow{\tau} T(m, \varphi_j) \text{ for some } j, 1 \le j \le k \text{ s.t. } m \vDash \varphi_j
\end{aligned}
$$

For the models $\mathcal{A}$ and $\mathcal{B}$ in Figure 1 and the distinguishing property $\Phi = \langle propose \rangle (\langle accept \rangle \langle applyLaw \rangle \mathbf{t} \wedge \langle amend \rangle \mathbf{t})$, the proof-tree $\Pi_{\mathcal{A}}^{\Phi}$ is shown in Figure 5(a). Note the conjunction in the property is encoded as two separate branches.

A proof-tree for a property φ and model M can be depicted graphically by projecting it onto M, which highlights the portion of M covered by the proof. For example, Figure 1(d) depicts, using dashed lines, the projection of the proof-tree shown in Figure 5(a) onto model $\mathcal{A}$.

Definition 12 (Proof-tree projection) *Let MTS $M = \langle S_M, Act, \Delta_M^r, \Delta_M^{\mathsf{p}}, m_0 \rangle$ and an $\mathcal{L}_\mu^{\mathsf{p}}$ property φ such that $M \vDash \varphi$ be given. Let Π_M^{φ} be a proof-tree for φ in M, defined as in Definition 11. Then* projection *of Π_M^{φ} is an MTS $N = \langle S_M, Act, \Delta_N^r, \Delta_N^{\mathsf{p}}, m_0 \rangle$ such that $\Delta_N^r \subseteq \Delta_M^r$ is the largest set where $(m \xrightarrow{\ell}_r m' \in \Delta_N^r)$ iff $((m, \varphi_0) \xrightarrow{\ell} (m', \varphi_1) \in \Pi_M^{\varphi})$, and $\Delta_N^{\mathsf{p}} \subseteq \Delta_M^{\mathsf{p}}$ is the largest set where $m \xrightarrow{\ell}_{\mathsf{p}} m' \in \Delta_N^{\mathsf{p}}$ iff $((m, \varphi_0) \xrightarrow{\ell} (m', \varphi_1) \in \Pi_M^{\varphi})$.*

To aid user comprehension, we explicitly display the ℓ transitions that are not possible from the leaf node of the form $\langle m, [\ell]\varphi \rangle$. For example, the proof-tree shown in Figure 5(b) has a leaf labelled $(4, [applyLaw]\mathbf{f})$ which indicates that there is no *applyLaw* transition from state 4 of model $\mathcal{B}$. This fact is shown in the projection of the proof-tree onto $\mathcal{B}$ in Figure 1(e) with a dashed crossed out transition. We call such augmented projections *explanations*. In Section 6 we show how to a modeler can use these explanations to explore the inconsistencies between two models.
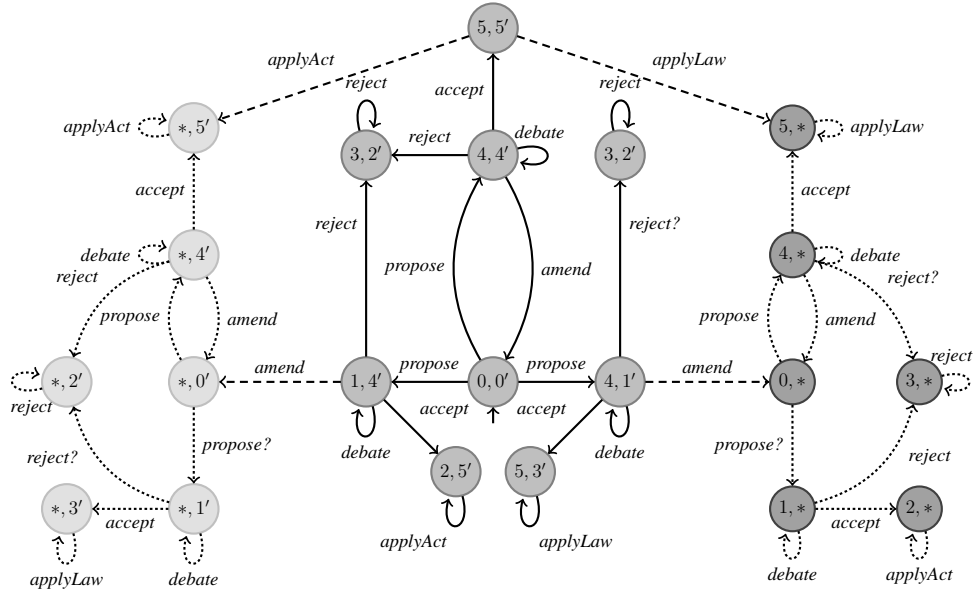
Fig. 6 Pseudo-merge of models $\mathcal{A}$ and $\mathcal{B}$. State $3, 2'$ appears twice for clarity.

5 Characterization of Inconsistency

In this section, we define *pseudo-merge* which is the basis for providing the various features described in Section 2. Intuitively, the pseudo-merge of two models is a model which distinguishes the behaviour for which the models agree and captures the states in which they show disagreement. We then provide an algorithm that builds a pseudo-merge that is correct and complete with respect to explaining the inconsistencies of the models being compared. These results allow us to conclude that pseudo-merge is a sound, compact representation of the inconsistencies between two models which can be used to generate appropriate feedback to users. In Section 6, we discuss how users can choose which of the many properties encoded in the pseudo-merge is to be used as feedback.

5.1 Pseudo-merge

A *pseudo-merge* of MTSs M and N is an MTS with two identified subsets of states: disagreement states for M (called D_M) and disagreement states for N (called D_N) such that if disagreement states for M are removed, then the pseudo-merge is a refinement of M, and if disagreement states for N are removed, then the pseudo-merge is a refinement of N. All other states are called *agreement* states, with the interpretation that the transitions between such states are behaviours that M and N agree upon. All behaviours leading to disagreement states are inconsistent with either M or N. Transitions between two disagreement states are called *disagreement transitions* while transitions that go from agreement states to disagreement states are called *boundary disagreement transitions*. The later are of

particular interest as they represent the first points in which one model disagrees with the other. Finally, a pseudo-merge for M and N with empty disagreement states is a common refinement of M and N.

Definition 13 (Pseudo-merge) *Let MTSs* $M = \langle S_M, Act, \Delta_M^r, \Delta_M^p, m_0 \rangle$ *and* $N = \langle S_N, Act, \Delta_N^r, \Delta_N^p, n_0 \rangle$ *be given. Their* pseudo-merge *is a tuple* $\langle P, D_M, D_N \rangle$, *where* $P = \langle S, Act, \Delta^r, \Delta^p, p_0 \rangle$ *is an MTS,* $D_M \subseteq S$ *and* $D_N \subseteq S$ *such that subMTSs of* P *over states* $S \setminus D_M$ *and* $S \setminus D_N$ *refine* M *and* N, *respectively. We call transitions in* $S \times Act \times (D_M \cup D_N)$ *and* $(S \setminus (D_M \cup D_N)) \times Act \times (D_M \cup D_N)$ disagreement transitions *and* boundary disagreement transitions, *respectively.*

A pseudo-merge of models $\mathcal{A}$ and $\mathcal{B}$ in Figures 1(a) and (b) is depicted in Figure 6. The disagreement states for $\mathcal{A}$ are labelled pairs in which the first element is $*$ (they appear on the left-hand side of Figure 6 and are shown in light grey). Similarly, the disagreement states for $\mathcal{B}$ have $*$ as as second element of the pair (they appear on the right-hand side of Figure 6 and are shown in dark grey). Boundary disagreement transitions are dashed, and the rest of the disagreement transitions are dotted.

We now present an algorithm, $+_{pm}$, for computing a pseudo-merge of two models. It is an adaptation of the algorithm introduced in [UC04] for constructing common refinements of consistent MTSs. The algorithm, applied to models $M = \langle S_M, Act, \Delta_M^r, \Delta_M^p, m_0 \rangle$ and $N = \langle S_N, Act, \Delta_N^r, \Delta_N^p, n_0 \rangle$, first builds a synchronous product by constructing the Cartesian product of $(S_M \cup \{*\}) \times (S_N \cup \{*\})$, where $*$ is a special symbol for denoting states in one model that do not have correspondences in the other. Second, the algorithm removes transitions related to specific non-deterministic choices from the result. Finally, the sets of disagreement states are defined as $D_N = (S_M \times \{*\})$ and $D_M = (\{*\} \times S_N)$. Intuitively, D_M means that M has gone into an inconsistent state from following N.

We now explain the algorithm in more detail. The synchronous product is built over the Cartesian product of $(S_M \cup \{*\}) \times (S_N \cup \{*\})$. The algorithm adds transitions resulting from executing M and N synchronously, i.e., simultaneous transitions synchronizing on the action labelling these transitions. Transitions in the synchronous product are computed based on the rules in Figure 7. For instance, if M and N can transit on ℓ through a required transition, then the synchronous product of M and N can transit on ℓ through a required transition as well, as indicated by rule **RR**. If M can transit on ℓ over a required transition and N can transit on ℓ over a maybe transition, this means that M has more information over the occurrence of ℓ than N does (N does not rule out the fact that that the transition on ℓ can become required or prohibited in the future). Hence, there is an agreement, and the synchronous product of M and N can transit on ℓ through a required transition. This is codified in rule **RM**, while the symmetric situation is described in rule **MR**. Using a similar reasoning, it is expected that the rules do not allow a transition on ℓ in the synchronous product if one model can transit on ℓ with a maybe transition while the other cannot transit on ℓ at all. The rules also do not allow an ℓ transition on the synchronous product of M and N if they both agree on prohibiting ℓ transitions.

Rules **FR** and **RF** are of particular interest as they capture the situation in which M and N disagree. For instance, **RF** states that if an ℓ-transition is required in M but prohibited in N, then the synchronous product has transitions on ℓ to a state in $S_M \times \{*\}$. Such states express that N has flagged the fact that a transition has occurred which is inconsistent with its own behaviour. Rules $\mathbf{\Gamma}*$ and $*\mathbf{\Gamma}$ encode the synchronous product once one of the models has reached a $*$-state. Essentially, the synchronous product allows the model that is not in a $*$-state to transition freely while prohibiting any transitions of the other model. The rules ensure that while one of the models has gone into a $*$-state, the synchronous product can simulate the behaviour of the other.

We give special treatment to the case in which both models have maybe ℓ-transitions: Rule **MM** states that the synchronous product of M and N has a maybe ℓ transition if M and N have maybe ℓ-transitions, and the states reached by these transitions are consistent (recall Definition 9). The intuition here is that we are not interested in introducing inconsistent pairs of states reachable through *maybe* transitions because these do not represent true disagreements between M and N as they can be removed by refining the *maybe* transition.

Once the synchronous product is constructed, a subset of transitions is then removed to resolve non-deterministic choices of M and N according to the following rule: if a state (m, n) of the synchronous product has an ℓ-transition to (m', n') such that m' and n' are inconsistent, remove this transition unless (i) $m \xrightarrow{\ell}_{\mathsf{r}} m'$ and there is no transition on ℓ from n to any state n'', where m' and n'' are consistent; or (ii) a dual case involving $n \xrightarrow{\ell}_{\mathsf{r}} n'$ occurs. The intuition for this rule is similar to that of rule **MM**: we do not want to include transitions to pairs of inconsistent states if these can be avoided in common implementations of M and N, because then they do not represent proper disagreements. The rule states that non-deterministic transitions on ℓ should be paired in a way that the resulting state in the synchronous product is consistent, if possible.

Definition 14 (The Synchronous Product) *Let* $M = \langle S_M, Act, \Delta_M^r, \Delta_M^p, m_0 \rangle$ *and* $N = \langle S_N, Act, \Delta_N^r, \Delta_N^p, n_0 \rangle$ *be MTSs. A* synchronous product *of M and N is an MTS* $P = \langle S_P, Act, \Delta_P^r, \Delta_P^p, p_0 \rangle$, *where* $S_P = S_M \times S_N$, $p_0 = (m_0, n_0)$, *and* Δ_P^r *and* Δ_P^p *are the smallest relations that satisfy the rules given in Figure 7.*

Definition 15 (The $+_{pm}$ Operator) *Let* M, N *and* P *be MTSs in Definition 14.* $M +_{pm} N = \langle S_P, Act, \Delta_{P'}^r, \Delta_{P'}^p, p_0 \rangle$, *where* $\Delta_{P'}^p$ *is as shown in Figure 8 and* $\Delta_{P'}^r = \Delta_P^r \cap \Delta_{P'}^p$.

Applying $+_{pm}$ to the models of the law-making process (see Figure 1 in Section 2), we obtain the model in Figure 6. This model includes three branches from the initial state on action *propose*, corresponding to all possible matchings containing at least one required transition of the non-deterministic choice on this action in the original models. The boundary disagreement transitions in these models are $(4, 1') \xrightarrow{amend} (4, *)$, $(5, 5') \xrightarrow{applyLaw} (5, *)$, $(5, 5') \xrightarrow{applyAct} (*, 5')$, and $(1, 4') \xrightarrow{amend} (*, 4')$.

As in the merge operation in [UC04], the pseudo-merge operator $+_{pm}$ works in $O(|M| \cdot |N| \cdot |Act|)$, where M and N are the original MTSs, size of a model is a number of states in it, and Act is their shared alphabet. The

$$\dfrac{m \xrightarrow{\ell}_{\mathsf{r}} m' \qquad n \xrightarrow{\ell}_{\mathsf{r}} n'}{(m,n) \xrightarrow{\ell}_{\mathsf{r}} (m',n')}\mathbf{RR} \qquad \dfrac{m \xrightarrow{\ell}_{\mathsf{m}} m' \qquad n \xrightarrow{\ell}_{\mathsf{m}} n' \qquad \mathrm{Cons}(m',n')}{(m,n) \xrightarrow{\ell}_{\mathsf{m}} (m',n')}\mathbf{MM}$$

$$\dfrac{m \xrightarrow{\ell}_{\mathsf{r}} m' \qquad n \xrightarrow{\ell}_{\mathsf{m}} n'}{(m,n) \xrightarrow{\ell}_{\mathsf{r}} (m',n')}\mathbf{RM} \qquad \dfrac{m \xrightarrow{\ell}_{\mathsf{m}} m' \qquad n \xrightarrow{\ell}_{\mathsf{r}} n'}{(m,n) \xrightarrow{\ell}_{\mathsf{r}} (m',n')}\mathbf{MR}$$

$$\dfrac{m \xrightarrow{\ell}_{\mathsf{r}} m' \qquad n \xrightarrow{\ell}{\not\longrightarrow}}{(m,n) \xrightarrow{\ell}_{\mathsf{r}} (m',*)}\mathbf{RF} \qquad \dfrac{m \xrightarrow{\ell}{\not\longrightarrow} \qquad n \xrightarrow{\ell}_{\mathsf{r}} n'}{(m,n) \xrightarrow{\ell}_{\mathsf{r}} (*,n')}\mathbf{FR}$$

$$\gamma \in \{\mathsf{r},\mathsf{m}\}\dfrac{m \xrightarrow{\ell}_{\gamma} m'}{(m,n) \xrightarrow{\ell}_{\gamma} (m',*)}\mathbf{\Gamma *} \qquad \gamma \in \{\mathsf{r},\mathsf{m}\}\dfrac{n \xrightarrow{\ell}_{\gamma} n'}{(*,n) \xrightarrow{\ell}_{\gamma} (*,n')}*\mathbf{\Gamma}$$

Fig. 7 Rules for the pseudo-merge operator $+_{pm}$.

$$\Delta^{\mathsf{p}}_{P'} = \Delta^{\mathsf{p}}_{P} \setminus \{(m,n) \xrightarrow{\ell}_{\mathsf{p}} (m',n') \in \Delta^{\mathsf{p}}_{P} \mid \neg\mathrm{Cons}(m',n') \wedge (m \xrightarrow{\ell}{\not\longrightarrow}_{\mathsf{r}} m' \vee \exists n'' \cdot (\mathrm{Cons}(m',n'') \wedge n \xrightarrow{\ell}_{\mathsf{p}} n''))$$
$$\wedge\ (n \xrightarrow{\ell}{\not\longrightarrow}_{\mathsf{r}} n' \vee \exists m'' \cdot (\mathrm{Cons}(m'',n') \wedge m \xrightarrow{\ell}_{\mathsf{p}} m''))\}$$

Fig. 8 Definition of possible transitions for $M +_{pm} N$.

maximal size of the resulting pseudo-merge is $(|M| + 1) \cdot (|N| + 1)$ since it is comprised of pairs of states in $M \cup \{*\} \times N \cup \{*\}$. We discuss the impact of the size of the pseudo-merge on the ability of users to inspect and select alternative explanations to an inconsistency in Section 6.

We shall now prove that the object built by $+_{pm}$ corresponds to the definition of a pseudo-merge which was tailored to capture refinement-related properties.

Theorem 2 *Let $M = \langle S_M, Act, \Delta^{\mathsf{r}}_M, \Delta^{\mathsf{p}}_M, m_0\rangle$ and $N = \langle S_N, Act, \Delta^{\mathsf{r}}_N, \Delta^{\mathsf{p}}_N, n_0\rangle$ be MTSs and let $M +_{pm} N$ be an MTS $P = \langle S_P, Act, \Delta^{\mathsf{r}}_P, \Delta^{\mathsf{p}}_P, p_0\rangle$, defined as in Definition 15. Then $\langle P, D_M, D_N\rangle$, where $D_N = (S_M \times \{*\})$ and $D_M = (\{*\} \times S_N)$, is a pseudo-merge of M and N.*

Proof Let P_M be a subMTS of P w.r.t. states $S_P \setminus D_M$. We show that $M \preceq P_M$ by defining the relation $\mathcal{R} = \{(m,(m,n))|(m,n) \in S_P \setminus D_M\}$ (n may well be $*$) and showing that $\mathcal{R}$ is a refinement relation between P_M and M; that is, it satisfies both conditions of Definition 7.

Let $((m,n),m) \in \mathcal{R}$. If $m \xrightarrow{\ell}_{\mathsf{r}} m'$, then by one of the rules **RR**, **RM**, **RF**, or $\mathbf{\Gamma *}$ with $\gamma = t$, there is a transition $(m,n) \xrightarrow{\ell}_{\mathsf{r}} (m',n')$. Of all such transitions, at least one has remained after the removal step shown in Figure 8. Otherwise, if a pair of consistent targets did not exist, the removed transitions could just be the ones corresponding to cases where neither $m \xrightarrow{\ell}_{\mathsf{r}} m'$ nor $n \xrightarrow{\ell}_{\mathsf{r}} n'$. These transitions would not have been inserted in the product in the first place because of the condition on rule **MM**. And since there is a pair of consistent targets,

$$((m,n),\ell,(m',n')) \in \Delta_G \Rightarrow (((\forall m'', ((m,n),\ell,(m'',n')) \in \Delta^{\mathsf{r}} \Rightarrow ((m,n),\ell,(m'',n')) \in \Delta_G) \wedge$$
$$(\forall n'', ((m,n),\ell,(m',n'')) \in \Delta_G \Rightarrow (n' = n'')) \wedge ((n,\ell,n') \in \Delta_N^{\mathsf{r}})) \vee$$
$$((\forall n'', ((m,n),\ell,(m',n'')) \in \Delta^{\mathsf{r}} \Rightarrow ((m,n),\ell,(m',n'')) \in \Delta_G) \wedge$$
$$(\forall m'', ((m,n),\ell,(m'',n')) \in \Delta_G \Rightarrow (m' = m'')) \wedge ((m,\ell,m') \in \Delta_M^{\mathsf{r}})))$$

Fig. 9 Formalization of condition (6) of Definition 16.

the corresponding transition has to remain in the product. On the other hand, the transition $(m,n) \xrightarrow{\ell}_{\mathsf{p}} (m',n')$ could not have been created by rules **FR** or $*\boldsymbol{\Gamma}$, since those would imply that $(m',n') \in D_M$. All other rules have the premise that $m \xrightarrow{\ell}_{\mathsf{p}} m'$; therefore, using $\mathcal{R}$ as a refinement relation proves the theorem. $\square$

5.2 Correctness and Completeness

We now define the set of $\mathcal{L}_\mu^p$ properties denoted by $M +_{pm} N$, show that all the properties in the set are distinguishing properties of M and N, and then that there are no distinguishing properties that provide shorter explanations (in the sense of Definition 12) of the inconsistency between M and N.

The set of $\mathcal{L}_\mu^p$ properties denoted by $M +_{pm} N$ is defined as those that can be constructed by any *distinguishing DAG* embedded into $M +_{pm} N$.

Definition 16 (Distinguishing DAG) *Let* $M +_{pm} N = \langle\langle S, Act, \Delta^{\mathsf{r}}, \Delta^{\mathsf{p}}, p_0\rangle, D_M, D_N\rangle$. *We call a DAG* $G = (v_0, V, \Delta_G)$, *where* $V \subseteq S$, *a* distinguishing DAG *iff all of the following conditions hold:*

(1)   G has the same initial state, i.e., $v_0 = p_0$;

(2)   G contains only required transitions, i.e., $\Delta_G \subseteq \Delta^{\mathsf{r}}$;

(3)   G contains no transitions from a disagreement state , i.e., $p \xrightarrow{\ell} p' \in \Delta_G \Longrightarrow p \notin (D_M \cup D_N)$;

(4)   Leaf transitions are the only disagreements, i.e., $p \in V \wedge (\forall p' \in S \cdot p \xrightarrow{\ell} p' \notin \Delta_G) \Longrightarrow p \in D_M \cup D_N$;

(5)   All transitions from a given state are on the same symbol, i.e., $(p \xrightarrow{\ell} p' \in \Delta_G \wedge p \xrightarrow{\ell'} p'' \in \Delta_G) \Longrightarrow \ell = \ell'$;

(6)   Each transition corresponds to taking all transitions on a symbol in one original MTS and only one, required, transition in the other, as formalized in Figure 9.

A distinguishing DAG represents a joint execution in MTSs M and N that highlights a disagreement. Rule (1) expresses the fact that the execution starts in the initial state of both models. Rule (2) means that each step corresponds to a required transition in at least one of the models. Rules (3) and (4) express minimality, while rule (5) ensures that only a single execution is followed. Finally, rule (6) means that a required transition on a given action in one model is matched by all transitions on the same action in the other. An example distinguishing DAG for $M +_{pm} N$ is the subgraph which consists of transitions $(0,0') \xrightarrow{propose}_{\mathsf{r}} (4,4') \xrightarrow{accept}_{\mathsf{r}} (5,5') \xrightarrow{applyLaw}_{\mathsf{r}} (5,*)$ and $(0,0') \xrightarrow{propose}_{\mathsf{r}} (4,1') \xrightarrow{amend}_{\mathsf{r}} (0,*)$.

Definition 17 (Property of a Distinguishing DAG) *The* property *induced by a distinguishing DAG* $G = (v_0, V, \Delta_G)$

on the pseudo-merge $\langle\langle S, Act, \Delta^r, \Delta^p, (m_0, n_0)\rangle, D_M, D_N\rangle$ *is an* $\mathcal{L}^p_\mu$ *property* $\mathcal{F}_G$, *defined inductively as follows:*

(i)   *If* $((m, n), \ell, (m', *)) \in \Delta_G$, *then* $\mathcal{F}_G((m, n)) = \langle\ell\rangle \mathbf{t}$;

(ii) *If* $((m, n), \ell, (*, n')) \in \Delta_G$, *then* $\mathcal{F}_G((m, n)) = [\ell]\mathbf{f}$;

(iii) *If* $\forall i, 1 \leq i \leq k \cdot ((m, n), \ell, (m', n'_i)) \in \Delta_G$, *or if* $k = 1$ *and* $m \xrightarrow{\ell}_r m'$,

     then $\mathcal{F}_G((m, n)) = \langle\ell\rangle \bigwedge_{i=1}^{k} \mathcal{F}_G((m', n'_i))$;

(iv) *If* $\forall i, 1 \leq i \leq k \cdot ((m, n), \ell, (m'_i, n')) \in \Delta_G$, *or if* $k = 1$ *and* $m \not\xrightarrow{\ell}_r m'_1$

     then $\mathcal{F}_G((m, n)) = [\ell] \bigvee_{i=1}^{k} \mathcal{F}_G((m'_i, n'))$.

Note that the property built from the above definition is not symmetrically defined. Indeed, we chose it to be *true*

in model M and *false* in model N, as shown in the proof of Theorem 3. The property for the distinguishing DAG

in our example is $\Phi = \langle propose\rangle(\langle accept\rangle\langle applyLaw\rangle\mathbf{t} \wedge \langle amend\rangle\mathbf{t})$ which evaluates to *true* in the model $\mathcal{A}$ and to

false in the model $\mathcal{B}$. Projecting this DAG onto the model $\mathcal{A}$ in Figure 1(a), we obtain the diagram of Figure 1(d)

(see Section 2), where the dashed edges correspond to those covered by the DAG.

Definition 18 (Properties of $+_{pm}$) *Let MTSs M and N be given. The set of properties of a pseudo-merge of M and*

N, $M +_{pm} N$ are those induced by all distinguishing DAGs of $M +_{pm} N$.

Theorem 3 (Correctness of $+_{pm}$) *Let MTSs M and N be given and let $M +_{pm} N$ be their pseudo-merge. Then*

all properties of $M +_{pm} N$ are distinguishing properties of M and N.

The proof consists of proving the following lemma which states that the distinguishing properties of $M +_{pm} N$

hold in M and their negation holds in N.

Lemma 1 *Let $M = \langle S_M, Act, \Delta^r_M, \Delta^p_M, m_0\rangle$ and $N = \langle S_N, Act, \Delta^r_N, \Delta^p_N, n_0\rangle$ be two inconsistent MTSs. Let G*

be a distinguishing DAG of $M +_{pm} N$ and $\varphi = \mathcal{F}_G(m_0, n_0)$. Then $M \vDash \varphi \wedge N \vDash \neg\varphi$.

Proof 1. If G has only one transition, then it is between a regular state $(m, n) \in S_M \times S_N$ (in this case (m_0, n_0))

and a disagreement state (case (4) of Definition 16). This transition could only have been created by rules **RF** or

FR.

If rule **RF** was applied, then the transition is of the form $(m, n) \xrightarrow{\ell} (m', *)$. Hence, $\varphi = \langle\ell\rangle\mathbf{t}$, by case (i) of

Definition 17. As a premise of rule **RF**, there is a transition $m \xrightarrow{\ell}_r m'$, so $M \vDash \varphi$. Since we know that there is no

transition on ℓ from N, so $N \vDash \neg\varphi$.

If rule **FR** was applied, we have a dual situation. The property we obtain is $[\ell]\mathbf{f}$ (case (ii) of Definition 17),

which is true in states of M that do not have a transition on ℓ and false in those states of N that require one.

2. Suppose the lemma holds on all DAGs with up to height k. We prove it for a DAG G with height $k + 1$.

Consider all transitions stemming from the root (m, n) of G. By case (6) of Definition 16, we have two dual cases.

Suppose these transitions are on ℓ from (m, n) to states $(m', n'_1), \ldots, (m', n'_k)$. By case (iii) of Definition 17, the property φ is then $\langle \ell \rangle \bigwedge_{i=1}^{k} \varphi_i$, where φ_i is a shorthand for $\mathcal{F}_G((m', n'_i))$. By induction hypothesis, $\forall i \in \{1, \ldots, k\},' \vDash \varphi_i \wedge n'_i \vDash \neg \varphi_i$. In that case, we also know that there is a transition $m \stackrel{\ell}{\longrightarrow}_r m'$. Therefore, $M \vDash \varphi$. With rules **RR** and **RM**, the operator builds a transition $(m, n) \stackrel{\ell}{\longrightarrow}_r (m', n')$ for *every* possible transition on ℓ from n. Therefore, $\forall n \stackrel{n}{\longrightarrow}_p{}', \exists i \in \{1, \ldots, k\}$ such that $n' = n'_i$. Since for all possible targets on ℓ from n the conjunction of the φ_is does not hold, $N \vDash \neg \varphi$.

The dual case is treated with case (iv) of Definition 17.

All properties induced by the pseudo-merge are built from a distinguishing DAG, and are therefore *true* in M and *false* in N. $\square$

We now express the notion of completeness of the pseudo-merge constructed by $+_{pm}$. We say that $M +_{pm} N$ is *complete* in the sense that for any explanation of a distinguishing property for M and N, there is an explanation for a property derived from $M +_{pm} N$ which is a sub-graph of the former.

Theorem 4 (Completeness of $+_{pm}$) *Let M and N be inconsistent MTSs. For any distinguishing property φ and two proofs, Π_M^φ and $\Pi_N^{\neg \varphi}$, there exists a property φ' induced by $M +_{pm} N$, and proofs $\Pi_M^{\varphi'}$ and $\Pi_N^{\neg \varphi'}$ such that the projection of $\Pi_M^{\varphi'}$ on M is a subgraph, with the same initial state, of the projection of Π_M^{φ} on M; and similarly, the projection of $\Pi_N^{\neg \varphi'}$ on N is a subgraph of $\Pi_N^{\neg \varphi}$ on N.*

Proof (Outline) We start by combining the given proofs, Π_M^φ and $\Pi_N^{\neg \varphi}$, into one tree. We then collapse those nodes that correspond to loops in the pseudo-merge, and cut some superfluous subtrees: the ones corresponding to an execution that already reached an inconsistency or redundant explanation of an inconsistency for a pair of states. The result is a simpler tree, encoding the (smaller) proof of another property. We show that we can project this tree onto the pseudo-merge, and that the projection yields a distinguishing DAG. We also show that the property induced by the distinguishing DAG is the same as proved by the tree. Finally, we split the tree into two proof-trees, one for each model. The way the new tree is built ensures that the new proofs start at the same state as the original ones and are projected as subgraphs of the original trees. $\square$

A detailed proof of this theorem is given in Appendix A.

It is interesting to note that if MTSs M and N are consistent, then $M +_{pm} N$ yields a common refinement. Although procedures for constructing common refinements from consistent MTSs exist (e.g., [FU08]), this result indicates that pseudo-merge extends the theory of MTSs in a sound way as it works both for consistent and inconsistent cases.

Input: Inconsistent MTSs $M = \langle S_M, Act, \Delta^r_M, \Delta^p_M, m_0 \rangle$ and $N = \langle S_N, Act, \Delta^r_N, \Delta^p_N, n_0 \rangle$,

their pseudo-merge $P = M +_{pm} N = \langle S_P, Act, \Delta^r_P, \Delta^p_P, p_0 \rangle$,

a boundary disagreement transition $t = (s, \ell, s') \in \Delta^r_P \cap ((S_M \times S_N) \times Act \times (S_P \setminus (S_M \times S_N)))$.

1. Compute the shortest sequence of transitions $\pi = t_0, \ldots, t_n$ in P from p_0 to s.

2. Append t to π.

3. Perform a depth-first search for a distinguishing DAG G (see rules in Definition 16) such that π is included in G:

   a. Define a set I containing the transitions of π and an empty set J.

   b. Add τ, the next transition in I or in J if I is empty, to G. Stop if both are empty.

   c. Add all new transitions required to comply to the rules of Definition 16 to J.

4. If found, **RETURN** G. Else **ABORT**.

Fig. 10 Algorithm for computing user-guided feedback.

6 User-guided Feedback Generation

We now show how a compact representation of all explanations to inconsistency, constructed in Section 5, can be used by a modeler to explore the inconsistencies between two models. Having constructed the pseudo-merge, the modeler can select a boundary disagreement transition asking why the models are inconsistent at that point. We show how this user-selected boundary disagreement transition can get converted into a distinguishing property that, when explained using the technique in Section 4, covers the selected transition.

The general algorithm for computing user-guided feedback on inconsistent models is shown in Figure 10. The algorithm receives the pseudo-merge of two models, M and N, and a boundary disagreement transition of the pseudo-merge, and produces a distinguishing DAG (see Definition 16) over the pseudo-merge which covers this transition. This disagreement DAG encodes (see Definition 17) a distinguishing property φ for M and N, i.e., there exist proof-trees showing that φ holds in M and does not hold in N, and, projected onto M and N respectively, these trees cover the boundary disagreement transition selected by the user. In other words, the algorithm constructs explanations, in the form of highlighted transitions in M and N, of a distinguishing property for M and N such that the explanations cover the user-selected disagreement transition.

The more detailed procedure is given in Appendix B.

Applying the algorithm on the pseudo-merge in Figure 6 and boundary disagreement transitions

$$\{(5, 5') \xrightarrow{applyLaw}_r (5, *), \ (4, 1') \xrightarrow{amend}_r (0, *), \ (5, 5') \xrightarrow{applyAct}_r (*, 5'), \ (1, 4') \xrightarrow{amend}_r (*, 0')\},$$

we respectively obtain distinguishing graphs corresponding to the properties

$$\langle propose\rangle(\langle accept\rangle\langle applyLaw\rangle\mathbf{t} \wedge \langle amend\rangle\mathbf{t})$$
$$\langle propose\rangle(\langle accept\rangle\langle applyLaw\rangle\mathbf{t} \wedge \langle amend\rangle\mathbf{t})$$
$$\langle propose\rangle(\langle accept\rangle[applyAct]\mathbf{f} \wedge \langle amend\rangle\mathbf{t})$$
$$[propose](\langle accept\rangle\langle applyLaw\rangle\mathbf{t} \vee [amend]\mathbf{f})$$

More specifically, the first boundary disagreement property $(5,5') \overset{applyLaw}{\longrightarrow}_{\mathsf{r}} (5,*)$ yields the following subset of the pseudo-merge of Figure 6:

$$(0,0') \overset{propose}{\longrightarrow} (4,1') \overset{amend}{\longrightarrow} (0,*); (0,0') \overset{propose}{\longrightarrow} (4,4') \overset{accept}{\longrightarrow} (5,5') \overset{applyLaw}{\longrightarrow} (5,*)$$

This DAG is then converted into formula Ψ and proof-trees of Figure 5. The first two properties are identical because both $(5,5') \overset{applyAct}{\longrightarrow}_{\mathsf{r}} (*,5')$ and $(4,1'), \overset{amend}{\longrightarrow}_{\mathsf{r}} (0,*)$ form part of exactly the same argument as to why the models are inconsistent.

There are cases when a distinguishing DAG covering the user-selected disagreement transition cannot be constructed. For example, suppose we want to distinguish between models $\mathcal{E}$ and $\mathcal{G}$ in Figure 11 and use their pseudo-merge, selecting transition $(3,2') \overset{d}{\longrightarrow}_{\mathsf{r}} (*,3')$. If we want to reach this transition, we have to take transition $(0,0') \overset{a}{\longrightarrow}_{\mathsf{r}} (2,1')$ first. However, since condition (6) of Definition 16 tells us to include all possible transitions on a from state 0, we have to add transition $(0,0') \overset{a}{\longrightarrow}_{\mathsf{r}} (1,1)$ to the DAG. Condition (4) of Definition 16 forces us to continue from $(1,1)$, which is not a disagreement state, including transition $(1,1) \overset{b}{\longrightarrow}_{\mathsf{r}} (0,0')$ which forms a loop, violating condition (1) that requires the graph to be acyclic.

The reason why we were unable to find a distinguishing DAG through $(3,2') \overset{d}{\longrightarrow}_{\mathsf{r}} (*,3')$ is that distinguishing properties that cover this transition are not *minimal*. For example, property $[a](\langle c\rangle[d]\mathbf{f}\vee\langle b\rangle\langle b\rangle\mathbf{t})$ covers the transition, but its proof-trees are strictly larger than those for the property $\langle b\rangle\mathbf{t}$. The latter are generated by the algorithm in Figure 10 when transition $(0,0) \overset{b}{\longrightarrow}_{\mathsf{r}} (4,*)$ is selected. We believe that the problem is caused by the rule for removing unnecessary transitions for non-deterministic case being too weak, and thus this irrelevant transition is kept in the pseudo-merge. Informally, the fact that we can reduce a distinguishing property covering $(3,2') \overset{d}{\longrightarrow}_{\mathsf{r}} (*,3')$ into one that does not cover it, shows that this disagreement is not relevant.

For the cases in which a distinguishing DAG cannot be constructed to cover the user-selected disagreement transition, as described above, the user is forced to select a different disagreement transition. In future work, we aim to strengthen the rules defining pseudo-merge in order to avoid such situations.

One of the potential difficulties related to the complexity of constructing a pseudo-merge (see Section 5) to allow for user-selected generation of inconsistency feedback is the size of the pseudo-merge state space. Pseudo-merge models can be large; it is the price to pay for providing all possible explanations to an inter-model inconsistency in one compact representation. The problem can, however, be mitigated via tool support. We believe that existing tools for validating traditional behaviour models can be extended to support inspecting pseudo-merge models and
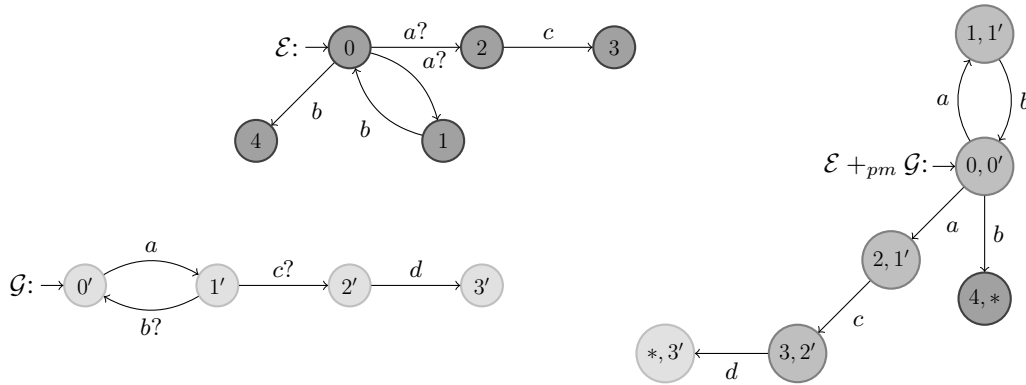
Fig. 11 Two models, $\mathcal{E}$ and $\mathcal{G}$, and their pseudo-merge $\mathcal{E} +_{pm} \mathcal{G}$.

selecting distinguishing DAGs. In particular, support for animation, hiding, minimization, and hierarchical states can aid these tasks significantly.

Summarizing, in this section we have shown how to build a distinguishing DAG that covers a user-selected disagreement transition in a pseudo-merge of MTSs M and N. The resulting DAG can be used to generate a distinguishing property that holds in M. The proof of such property in M and the proof of its negation in N can be visualized as explanations (see Section 4). This process allows us to provide sound feedback on the inconsistencies between the models being compared. We have also shown that the pseudo-merge may include some disagreement transitions, related to non-deterministic choices, that do not yield minimal distinguishing properties. Although this situation did not occur in the validation of our approach, the practical implications, if it does occur, is that the user, when selecting one of these transitions, may be required to pick a different transition in order to produce feedback on inconsistency.

7 Validation

In this section, we briefly describe tool support for our approach and then discuss two case studies.

7.1 Tool Support

In order to validate our approach, we have built a prototype tool [Sas09] that constructs a pseudo-merge from two MTS models and an $\mathcal{L}_\mu^p$ property for a given boundary disagreement transition, when possible. The tool is implemented in OCaml, and exchanges information with MTSA [DFCU08] by using the same textual representation for MTSs in order to allow automated graphical representation of models.

– The system: a printer with a pre-output tray

– One model:

  – No difference between input and output

  – No modelling of the repairman or the job giver/taker

– Alphabet :

  idle The printer doesn't do anything

  jobIn A print job is given to the printer

  jobOut Ejects the printed job and reports

  jobDiscard Discard whatever was printed and reports

  takeSheet The rolls takes a sheet from the tray

  printSheet The ink is put on the paper

  staple Staple the sheets together

block The sheet folds and get stuck in the printer

noPaper No more paper in the tray

noInk No more ink in the toner

resume A repairman fixes the printer

– Requirement: A sheet never goes backwards. Instead, it follows these steps:

  1. sheet is on the paper tray

  2. sheet is taken off the tray

  3. sheet is printed

  4. sheet goes on the pre-output tray

  5. sheet goes on the output tray (or garbage)

Table 1 The informal specification of the printer.

7.2 The Printer Case Study

The aim of this case study was to (1) compare models developed by people not involved in the development of our approach; and (2) to evaluate the effectiveness of our techniques to identify and explain the inconsistencies between the models.

We developed a deliberately underspecified natural language specification of a printer controller (see Table 1), and two PhD students with behaviour modelling experience were requested to build behaviour models from the specification. Our approach was then used to generate examples of inconsistency which became the basis of a "negotiation" – discussion with the two modellers aimed to understand the difference in their interpretation of the printer specification.

The specification fixed a communication alphabet for the printer controller and described the process for printing paper (from the paper tray through to the output tray), requiring that the paper never goes backwards even in the case of events such as *noInk*, *jobDiscard*, (paper-)*block*. Note that we asked the modellers to produce LTS rather than MTS models (i.e., to make explicit decisions when they encounter underspecification, since LTSs do not allow possible transitions). The goal here was two-fold: to eliminate problems stemming from inexperience with the MTS formalism, and to increase the likelihood of modellers introducing inconsistencies.

Figure 12 shows the resulting models. The pseudo-merge, depicted in Figure 13, results in 13 boundary disagreement transitions, from which 11 different distinguishing properties were constructed. Interestingly, although model $\mathcal{H}$ in Figure 12(a) is non-deterministic, all properties were linear, and hence feedback was produced in the
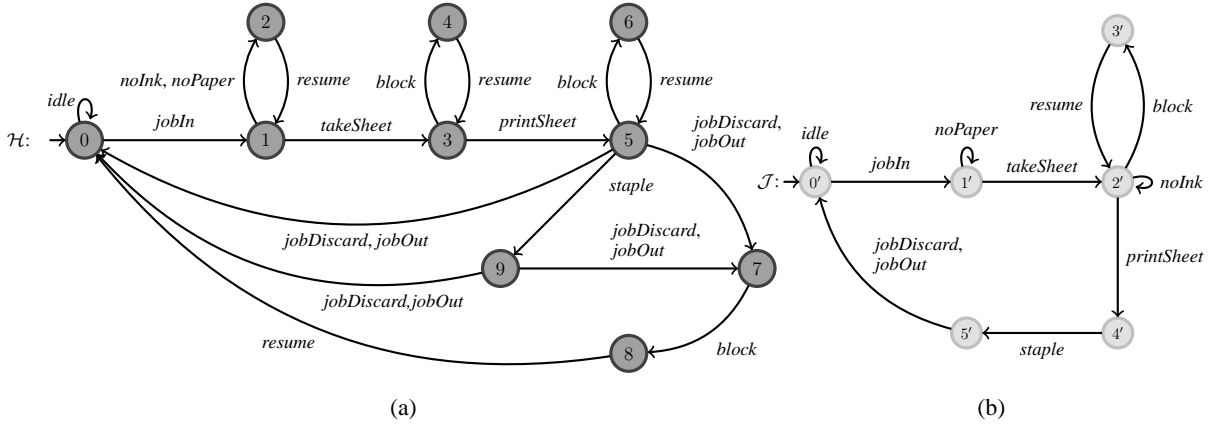
Fig. 12 Two models of a printer.

form of traces. This is because the non-deterministic behaviour in model $\mathcal{H}$ was matched by deterministic behaviour in model $\mathcal{J}$ (see Figure 12(b)).

Trace *jobIn*, *takeSheet*, *printSheet*, *jobOut* is an example of an inconsistency produced by our approach. It identifies a difference in criteria for when stapling occurred: in model $\mathcal{H}$, it was an optional step in the workflow, while model $\mathcal{J}$ made it mandatory. The trace corresponds to the proof of the property

$$\langle jobIn \rangle \langle takeSheet \rangle \langle printSheet \rangle \langle jobOut \rangle \mathbf{t}$$

which holds in model $\mathcal{H}$. A second example trace, *jobIn*, *noPaper*, *takeSheet*, identifies an inconsistency on a completely unrelated matter: in model $\mathcal{H}$, human intervention, followed by pressing the *resume* button, is needed to handle errors such as the lack of paper, whereas in the other model, the printer can sense that paper has been introduced and can *takeSheet* immediately. This trace corresponds to the proof of the distinguishing property $\langle jobIn \rangle \langle noPaper \rangle \langle takeSheet \rangle \mathbf{t}$ over model $\mathcal{J}$; its negation holds in model $\mathcal{H}$.

7.3 The Safety Injection System Case Study

In this study, we explored inconsistencies between behavior models automatically synthesized from declarative specifications. The inconsistent models were generated by making changes to the declarative specification, and our goal was to validate whether exploring the pseudo-merge could generate examples traceable back to the changes we made in the specification.

The safety injection system is part of a controller for a nuclear power plant. It is intended to maintain a sufficiently high pressure of coolant in the reactor, except when it operates in a special *Overridden* mode, in which case a lower pressure is allowed. Discrete time is modelled explicitly through action *tick*. When it is *enable*d, the controller receives information on the current pressure level inside the plant. It can then send signals to the plant to
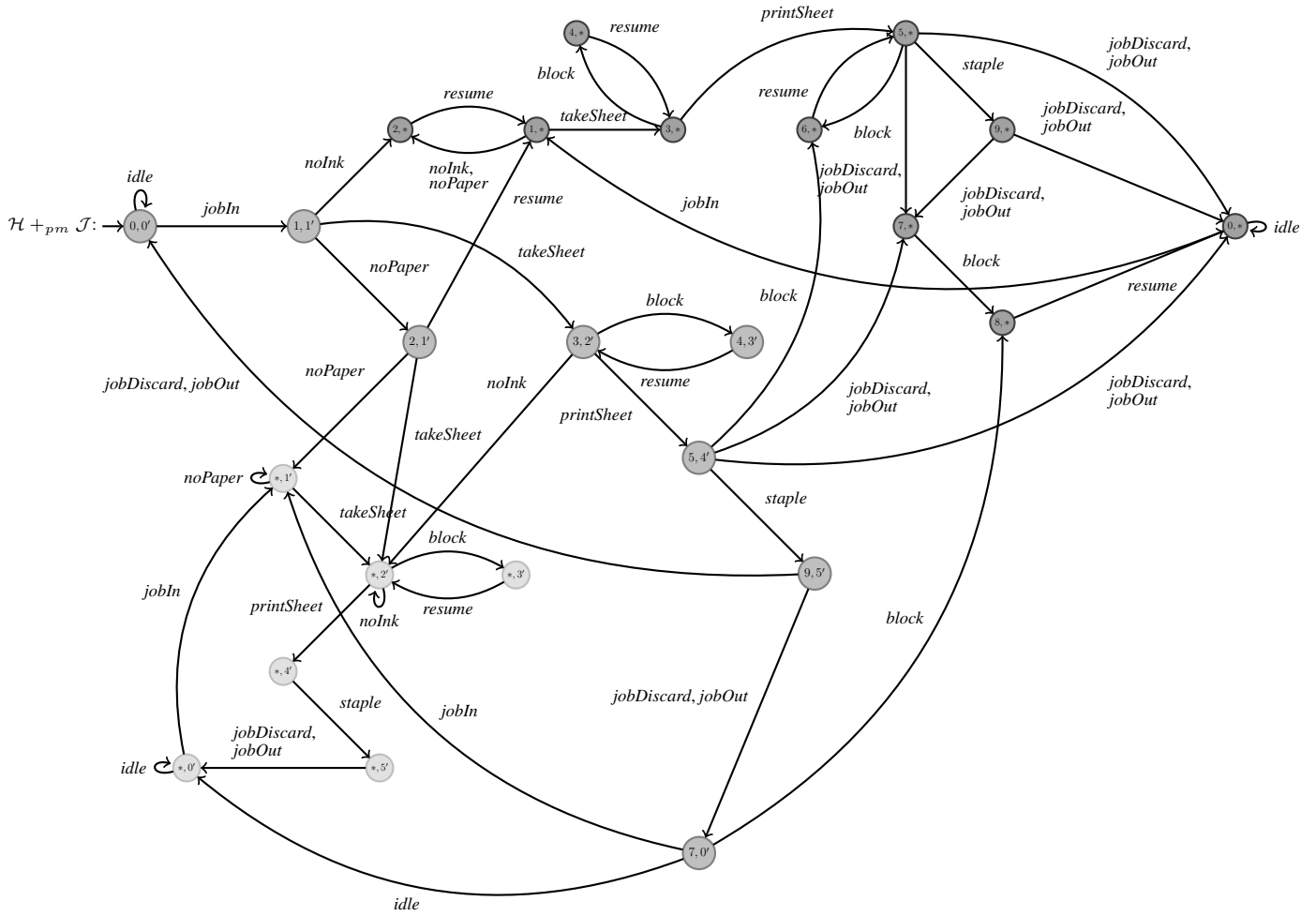
Fig. 13 The pseudo-merge of the printer models shown in Figure 12.

start (resp. stop) the injection system through action *sendSignal* (resp. *stopSignal*). The specification is given as a set of pre-, post- and trigger-conditions (formalized in fluent linear temporal logic [GM03]) over the operations to be provided by the system. This model was reported in [LKMU08].

Two behaviour models were automatically synthesized [LKMU08]: one from the original and the other from a modified specification. The modifications consisted of changing the pre- and the triggering conditions related to the urgency for starting and stopping the safety injection signal. Rather than requiring these actions to occur within one time unit after sensing particular environment conditions, we required them to be triggered in the *next* time unit. In addition, to increase the degree of non-determinism in the models, we abstracted away the various degrees of domain-related quantities in the model. For instance, rather than having events modelling the change of pressure levels to x (*raisePressure[x]* and *lowerPressure[x]*), we introduced more abstract events indicating whether the

pressure level was raised or lowered w.r.t. the relevant threshold. The resulting synthesized models have 36 and 48 states, respectively.

The pseudo-merge of these models consists of 220 states and can be animated in order to aid the exploration of agreement and disagreement states. Of the 104 disagreement transitions, one of the distinguishing properties we have found was

$$\Psi = \langle enable \rangle \langle tick \rangle \langle pressure \rangle (\langle sendSignal \rangle \langle tick \rangle \langle stopSignal \rangle \langle tick \rangle \mathbf{t} \;\wedge\; [tick]\mathbf{f}).$$

When projected on the synthesized models, this feedback directly relates to the changes made in the specification (due to space restrictions, we cannot show this projection). The feedback indicates that once the safety injection has been enabled and there is high pressure, the two models disagree on the potential behaviour of the system. In one model, it is possible to send and then stop sending the safety injection signal (the $\langle sendSignal \rangle \langle tick \rangle \langle stopSignal \rangle \langle tick \rangle \mathbf{t}$ branch), and time cannot advance (the $[tick]\mathbf{f}$ branch) without the controller performing an action. This relates to the urgency requirement for sending and stopping safety injection signals that was part of the original specification. In the other model, either sending and stopping the signal is not possible, or time can advance without the controller performing an action. This relates to the change in the urgency requirement: in the modified specification, sending and stopping the signal must occur after one time unit, i.e., after the occurrence of a *tick*.

8 Related Work

Generation and Analysis of Counterexamples. The original counterexample generation algorithm [CGMZ95], implemented in most symbolic model-checkers, produces linear counterexamples. It was extended to handle arbitrary ACTL properties [CLJV02] using the notion of *tree-like* counterexamples. However, navigating to "interesting" parts of the counterexamples, and thus understanding them, remains difficult.

Our work assumed that proofs of whether a μ-calculus property holds or fails in the model can be obtained, i.e., using the techniques of [Nam01, TC02], which concentrate not only on creation of the proof but also on techniques for presenting it to the user. We have relied on being able to overlay the proof onto the original models and then use animation to help present the evidence to the user. More sophisticated methods for evidence presentation, which provide a variety of graphical views, have been developed as well, e.g. [DRS03]. We also found it essential to be able to generate multiple causes of inconsistency, which are encoded compactly in our pseudo-merge. Multiple counterexamples have been generated in the context of LTL by [CIW+03] and in the context of CTL by [CG07]. In both approaches, users can visualize the result in various ways.

The problem of the automatic analysis of counterexamples was addressed by many researchers, e.g., [GV03, BPR03]. While we assume that the (human) modeller does the analysis, our work is complementary to this line of research.

Treatment of Inconsistency. A number of approaches to inconsistency management have been studied in the context of viewpoint-based modeling [NKF94]. Some of this work, e.g., [FGH+94, NCEF02], detects inconsistencies by using first-order logic rules and does not consider merge as a means of model exploration and inconsistency detection. Other researchers [HP01, EC01, SE03, NC05] propose ways of merging viewpoint models, where inconsistency is either explicitly represented using multi-valued logic [EC01, SE03, NC05] or resolved across inconsistent viewpoints by using a dominance ordering on owners of the viewpoints [HP01].

Our work is similar in spirit to [NC05], but our goal is not yet to support negotiation, but just help users identify causes of their disagreement. We augment the work of [NC05] with proof generation and visualization techniques.

Our pseudo-merge effectively allows to represent inconsistencies, and is thus similar to the approaches of [EC01, SE03]. In contrast, the work of [NSC+07] represents inconsistencies as variabilities, assuming that the disagreements have already been resolved, and thus any remaining discrepancies should be treated as variabilities in the system's intended functionality.

9 Conclusion and Future Work

In this paper, we have presented a well-founded approach to providing feedback on inconsistencies between partial behaviour models expressed as MTSs; a special case of this approach is providing feedback for non-bisimilar LTS models. We have shown why feedback in the form of traces is not adequate in the general case and how sound explanations for inconsistencies, derived from formal proofs of distinguishing properties, can be visualized as branching structures, depicted independently or overlaid on the models being compared. We have shown how the pseudo-merge, implemented using the $+_{pm}$ operator, compactly represents all relevant explanations of inconsistency between two models. Pseudo-merges can be used by modelers to guide the generation of explanations by selecting disagreement transitions, in order to explore causes of inconsistencies.

While it has not hindered the application of our approach so far, we are planning to address the problem of dealing with disagreement transitions in the pseudo-merge that do not represent distinguishing properties with minimal explanations. We believe that this can be accomplished by refining the notion of pseudo-merge.

The scalability of our approach is currently limited by the support to inspect large pseudo-merge models and the distinguishing properties constructed from them. Although a large pseudo-merge model is the price to pay for providing feedback on all possible explanations to an inter-model inconsistency in one compact representation, scalability, however, can be achieved via tool support. We believe that tool support for validating traditional behaviour models can be extended to support validating pseudo-merge models. In particular, we envisage the need for animation, hiding, minimization, and hierarchical states. We aim to integrate our prototype into the MTSA toolset which provides some of this functionality and use it to further validate the approach.

Acknowledgements

References

[BCU06]   G. Brunet, M. Chechik, and S. Uchitel. "Properties of Behavioural Model Merging". In *Proceedings of International Conference on Formal Methods (FM'06)*, volume 4085 of *LNCS*, pages 98–114. Springer, August 2006.

[BPR03]   T. Ball, A. Podelski, and S. Rajamani. "Boolean and Cartesian Abstraction for Model Checking C Programs". *International Journal of Software Tools for Technology Transfer (STTT)*, 5(1):49–58, 2003.

[CCGR99] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. "NUSMV: a new Symbolic Model Verifier". In *Proceedings of CAV'99*, volume 1633 of *LNCS*, pages 495–499, 1999.

[CG07]   M. Chechik and A. Gurfinkel. "A Framework for Counterexample Generation and Exploration". *International Journal of Software Tools for Technology Transfer (STTT)*, 9(5-6), 2007.

[CGMZ95] E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. "Efficient Generation of Counterexamples and Witnesses in Symbolic Model Checking". In *Proceedings of 32nd Design Automation Conference (DAC'95)*, pages 427–432, 1995.

[CGP99]   E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[CIW$^+$03] F. Copty, A. Irron, O. Weissberg, N. Kropp, and G. Kamhi. "Efficient Debugging in a Formal Verification Environment". *International Journal on Software Tools for Technology Transfer (STTT)*, 4(3):335–348, May 2003.

[CLJV02]   E.M. Clarke, Y. Lu, S. Jha, and H. Veith. "Tree-Like Counterexamples in Model Checking". In *Proceedings of the Seventeenth Annual IEEE Symposium on Logic in Computer Science (LICS'02)*, pages 19–29. IEEE Computer Society, July 2002.

[CPS93]   R. Cleaveland, J. Parrow, and B. Steffen. "The Concurrency Workbench: A Semantics Based Tool for the Verification of Concurrent Systems". *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[DFCU08] N. D'Ippolito, D. Fishbein, M. Chechik, and S. Uchitel. "MTSA: The Modal Transition System Analyzer". In *Proceedings of International Conference on Automated Software Engineering (ASE'08)*, pages 475–476, September 2008.

[DRS03]   Y. Dong, C.R. Ramakrishnan, and S. A. Smolka. "Evidence Explorer: A Tool for Exploring Model-Checking Proofs". In *Proceedings of the 15th International Conference on Computer-Aided Verification (CAV'03)*, volume 2725 of *LNCS*, pages 215–218, 2003.

[EC01]   S. Easterbrook and M. Chechik. "A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints". In *Proceedings of International Conference on Software Engineering (ICSE'01)*, pages 411–420. IEEE Computer Society Press, May 2001.

[FGH+94] A. C. W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. "Inconsistency Handling in Multi-Perspective Specifications". *IEEE Transactions on Software Engineering*, 20(8):569–578, 1994.

[FU08]   D. Fischbein and S. Uchitel. "On Correct and Complete Merging of Partial Behaviour Models". In *Proceedings of SIGSOFT Conference on Foundations of Software Engineering (FSE'08)*, pages 297–307, November 2008.

[GM03]   D. Giannakopoulou and J. Magee. "Fluent Model Checking for Event-Based Systems". In *Proceedings of the 9th joint meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'03)*, pages 257–266. ACM Press, September 2003.

[GV03]   A. Groce and W. Visser. "What Went Wrong: Explaining Counterexamples". In *Proceedings of SPIN Workshop on Model Checking of Software*, pages 121–135, 2003.

[Har87]   D. Harel. "StateCharts: A Visual Formalism for Complex Systems". *Science of Computer Programming*, 8:231–274, 1987.

[HJS01]   M. Huth, R. Jagadeesan, and D. A. Schmidt. "Modal Transition Systems: A Foundation for Three-Valued Program Analysis". In *Proceedings of 10th European Symposium on Programming (ESOP'01)*, volume 2028 of *LNCS*, pages 155–169. Springer, 2001.

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

[Hol97]   G.J. Holzmann. "The Model Checker SPIN". *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[HP01]   M. Huth and S. Pradhan. "Model-Checking View-Based Partial Specifications". *Electronic Notes in Theoretical Computer Science*, 45, November 2001.

[Kel76]   R. Keller. "Formal Verification of Parallel Programs". *Communications of the ACM*, 19(7):371–384, 1976.

[LKMU08] E. Letier, J. Kramer, J. Magee, and S. Uchitel. "Deriving Event-Based Transition Systems from Goal-Oriented Requirements Models". *Journal of Automated Software Engineering*, 15(2):175–206, 2008.

[LT88]   K.G. Larsen and B. Thomsen. "A Modal Process Logic". In *Proceedings of 3rd Annual Symposium on Logic in Computer Science (LICS'88)*, pages 203–210. IEEE Computer Society Press, 1988.

[Mil80]   R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, 1980.

[Mil99]   R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

[Nam01]   K. Namjoshi. "Certifying Model Checkers". In *Proceedings of 13th International Conference on Computer-Aided Verification (CAV'01)*, volume 2102 of *LNCS*. Springer-Verlag, 2001.

[NC05]   S. Nejati and M. Chechik. "Let's Agree to Disagree". In *Proceedings of 20th IEEE International Conference on Automated Software Engineering (ASE'05)*, pages 287 – 290. IEEE Computer Society, 2005.

[NCEF02] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. "xlinkit: A Consistency Checking and Smart Link Generation Service". *ACM Transactions on Internet Technology*, 2(2):151–185, 2002.

[NKF94]   B. Nuseibeh, J. Kramer, and A. Finkelstein. "Framework for Expressing the Relationship Between Multiple Views in Requirements Specifications". *IEEE Transactions on Software Engineering*, 20(10):760–773, October 1994.

[NSC+07] S. Nejati, M. Sabetzdeh, M. Chechik, S. Easterbrook, and P. Zave. "Matching and Merging of Statecharts Specifications". In *Proceedings of the 29th International Conference on Software Engineering (ICSE'07)*, pages 54–64, May 2007.

[PB00]   A.W. Roscoe P. Broadfoot. "Tutorial on FDR and Its Applications". In *Proceedings of the 9th SPIN Workshop on Model Checking Software (SPIN'00)*, volume 1885 of *LNCS*, page 322. Springer, 2000.

[Sas09]   M. Sassolas. "PseudoMerge: a Prototype Tool". `http://pagesperso-systeme.lip6.fr/Mathieu.Sassolas/recherche/tools/Pseu%doMergePrototype.zip`, 2009.

[SE03]   M. Sabetzadeh and S.M. Easterbrook. "Analysis of Inconsistency in Graph-Based Viewpoints: A Category-Theoretic Approach". In *Proceedings of 18th IEEE International Conference on Automated Software Engineering (ASE'03)*, pages 12–21. IEEE Computer Society, October 2003.

[TC02]   L. Tan and R. Cleaveland. "Evidence-Based Model Checking". In *Proceedings of 14th Conference on Computer-Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 455–470. Springer-Verlag, July 2002.

[UBC09]   S. Uchitel, G. Brunet, and M. Chechik. "Synthesis of Partial Behavior Models from Properties and Scenarios". *IEEE Transactions on Software Engineering*, 35(3):384–406, 2009.

[UC04]   S. Uchitel and M. Chechik. "Merging Partial Behavioural Models". In *Proceedings of 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'04)*, pages 43–52, November 2004.

A Proof of completeness of the pseudo-merge

In this section, we prove Theorem 4.

The outline of the proof is as follows: We start by combining the given proofs into a single tree (Section A.1). We then collapse this tree, removing nodes that correspond to loops in the pseudo-merge and cutting superfluous subtrees (Section A.2). This results in another tree that encodes the proof of a new, smaller, formula. We show that we can project this tree onto the pseudo-merge, and that the projection is a distinguishing DAG (Section A.3). Moreover, we show that the distinguishing DAG induces the same formula as proved by the tree (Section A.4). Finally, we split the tree into two proof-trees, one for each model. Our construction ensures that the new proofs are projected as subgraphs of the original ones (Section A.5).

We illustrate steps of this proof using the following example.

Example.   Consider models $\mathcal{X}$ and $\mathcal{Y}$ in Figure 14. Suppose that the original distinguishing property is

$$\Lambda = \langle a \rangle ([b]\langle d \rangle \langle c \rangle \langle c \rangle [a][d]\mathbf{f} \wedge [c]\langle d \rangle \langle b \rangle [a]\mathbf{f})$$

Proof-trees for this property on the models are shown on Figure 15. We do not show the projections of these proof-trees on the original models since in both cases they correspond nearly to the entire respective model. In $\mathcal{Y}$, the only transition not included in the projection is the self-loop on b in state 4; in $\mathcal{X}$ it is the self-loop on b in state $6'$.

(a)

(b)
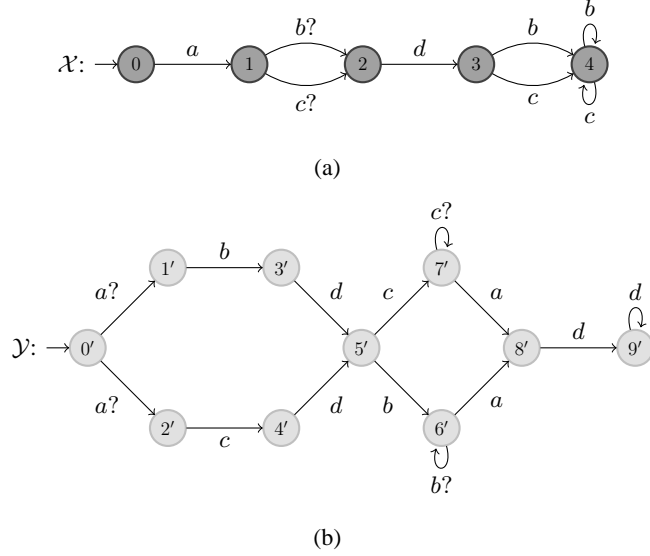
Fig. 14 Models used to illustrate completeness of the pseudo-merge.

A.1 Combining the proof-trees

First, we define a *combination* $\Pi^\varphi_{(M,N)}$ of the proof-trees Π^φ_M and $\Pi^{\neg\varphi}_N$. Intuitively, $\Pi^\varphi_{(M,N)}$ encodes the two very similar proofs for each model into a proof that φ is a distinguishing formula for M and N. This combination can be built with the $Comb$ operator.

Definition 19 (The $Comb$ operator) *Let M and N be MTSs and let $\varphi in \mathcal{L}^p_\mu$ be a distinguishing formula. Given two proof-trees Π^φ_M and $\Pi^{\neg\varphi}_N$ for φ in M and its negation in N, respectively, we define the $Comb$ operator inductively as follows:*

(i) $Comb((\Pi^\varphi_M, (*, \bot))) = \Pi^\varphi_M$, *where every node* (m, φ') *is replaced by a node* $((m, *), \varphi')$;

(ii) $Comb((*, \bot), (\Pi^{\neg\varphi}_N)) = \Pi^{\neg\varphi}_N$, *where every node* (n, φ') *is replaced by a node* $((*, n), \varphi')$;

(iii) $Comb((m, \langle\ell\rangle\varphi) \xrightarrow{\ell} (m', \varphi), (n, [\ell]\neg\varphi)) = ((m, n), \langle\ell\rangle\varphi) \xrightarrow{\ell} Comb((m, *), \varphi)$;

(iv) $Comb((m, [\ell]\varphi), (n, \langle\ell\rangle\neg\varphi) \xrightarrow{\ell} (n', \neg\varphi)) = ((m, n), [\ell]\varphi) \xrightarrow{\ell} Comb((*, n), \varphi)$;

(v) $Comb((m, \langle\ell\rangle\varphi) \xrightarrow{\ell} (m', \varphi), (n, [\ell]\neg\varphi) \xrightarrow{\ell} \{(n'_i, \neg\varphi)\}_i) = ((m, n), \bigwedge^k_{i=1} \varphi) \xrightarrow{\ell} \{Comb((m', \varphi), (n'_i, \neg\varphi))\}_i$;

(vi) $Comb((m, [\ell]\varphi) \xrightarrow{\ell} \{(m'_i, \varphi)\}_i, (n, \langle\ell\rangle\neg\varphi) \xrightarrow{\ell} (n', \neg\varphi)) = ((m, n), \bigvee^k_{i=1} \varphi) \xrightarrow{\ell} \{Comb((m'_i, \varphi), (n', \neg\varphi))\}_i$;

(vii) $Comb((m, \bigwedge^k_{i=1} \varphi_i) \xrightarrow{\tau} \{(m, \varphi_i)\}_i, (n, \bigvee^k_{i=1} \neg\varphi_i) \xrightarrow{\tau} (n, \neg\varphi_{i_0})) =$
$((m, n), \bigwedge^k_{i=1} \varphi_i) \xrightarrow{\tau} Comb((m, \varphi_{i_0}), (n, \neg\varphi_{i_0}))$;

(viii) $Comb((M, \bigvee^k_{i=1} \varphi_i) \xrightarrow{\tau} (M, \varphi_{i_0}), (N, \bigwedge^k_{i=1} \neg\varphi_i) \xrightarrow{\tau} \{(N, \neg\varphi_i)\}_i) =$
$((M, N), \bigvee^k_{i=1} \varphi_i) \xrightarrow{\tau} Comb((M, \varphi_{i_0}), (N, \neg\varphi_{i_0}))$

Rules (v) and (vi) of Definition 19 duplicate the subformula φ, but the obtained formula is logically equivalent to the original one. We treat each copy separately; in particular, we keep track of correspondences between each
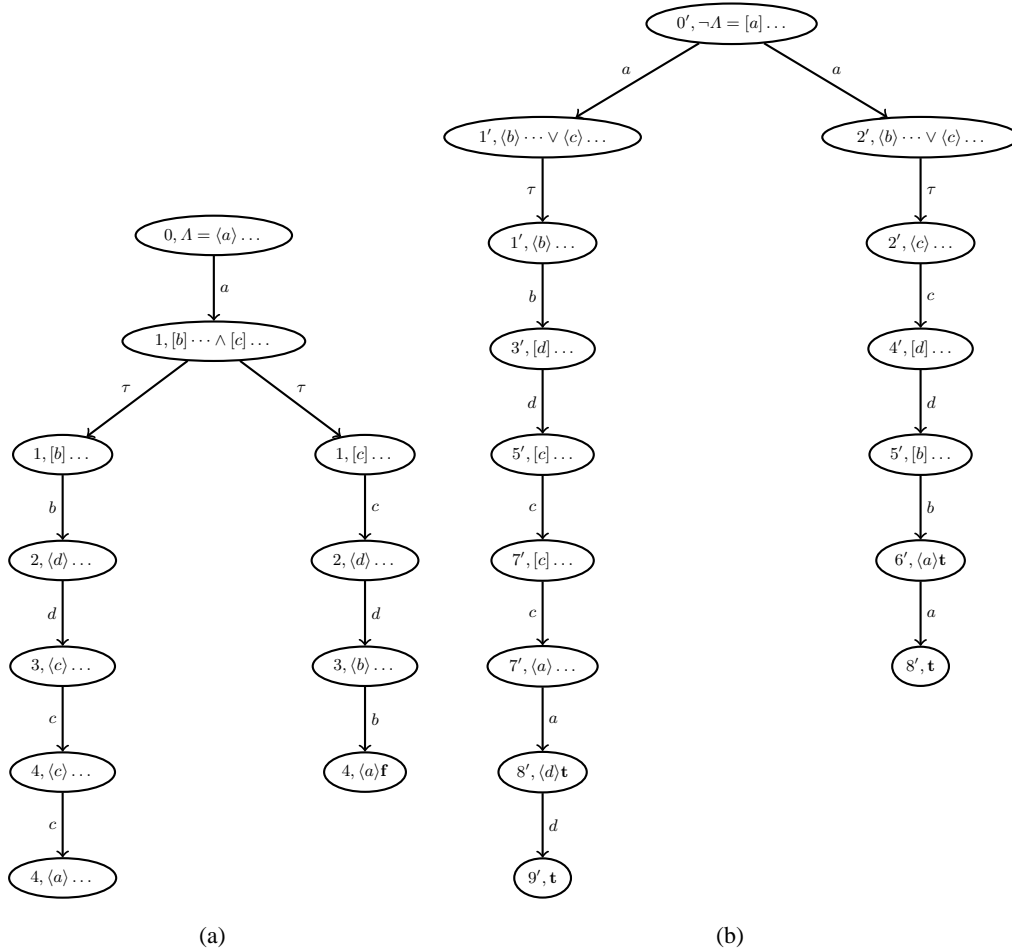
Fig. 15 The original proof-trees showing that Λ is *true* in model $\mathcal{X}$ and *false* in model $\mathcal{Y}$: (a) a proof-tree for $\mathcal{X} \vDash \Lambda$; (b) a proof-tree for $\mathcal{Y} \vDash \neg\Lambda$.

child and each duplicate. Note that we actually lose some information in cases (vii) and (viii), but it is either not important (in the sense that it represents a proof for a part of the formula that gets removed) or present in a sibling, and can be retrieved when needed. The main property of this combined tree is that in each node $((m,n), \varphi')$ with both $m \neq *$ and $n \neq *$, $m \vDash \varphi'$ and $n \vDash \neg\varphi'$. This property holds because for each such $((m,n), \varphi')$, there are nodes $(m, \varphi') \in \Pi_M^\varphi$ and $(n, \neg\varphi') \in \Pi_N^{\neg\varphi}$.

Example. Combining the proof-trees of Figure 15 yields the proof-tree in Figure 16(a). If we project it onto the pseudo-merge of models $\mathcal{X}$ and $\mathcal{Y}$, displayed on Figure 17, we get almost the entire graph (only the self-loop on b in state $(4, 6')$ is not included in the projection). Here, we do not have a DAG because of the loop on state $(4, 7')$ (hence the graph is not acyclic); furthermore, there are outgoing transitions on both b and c from state $(3, 5')$,
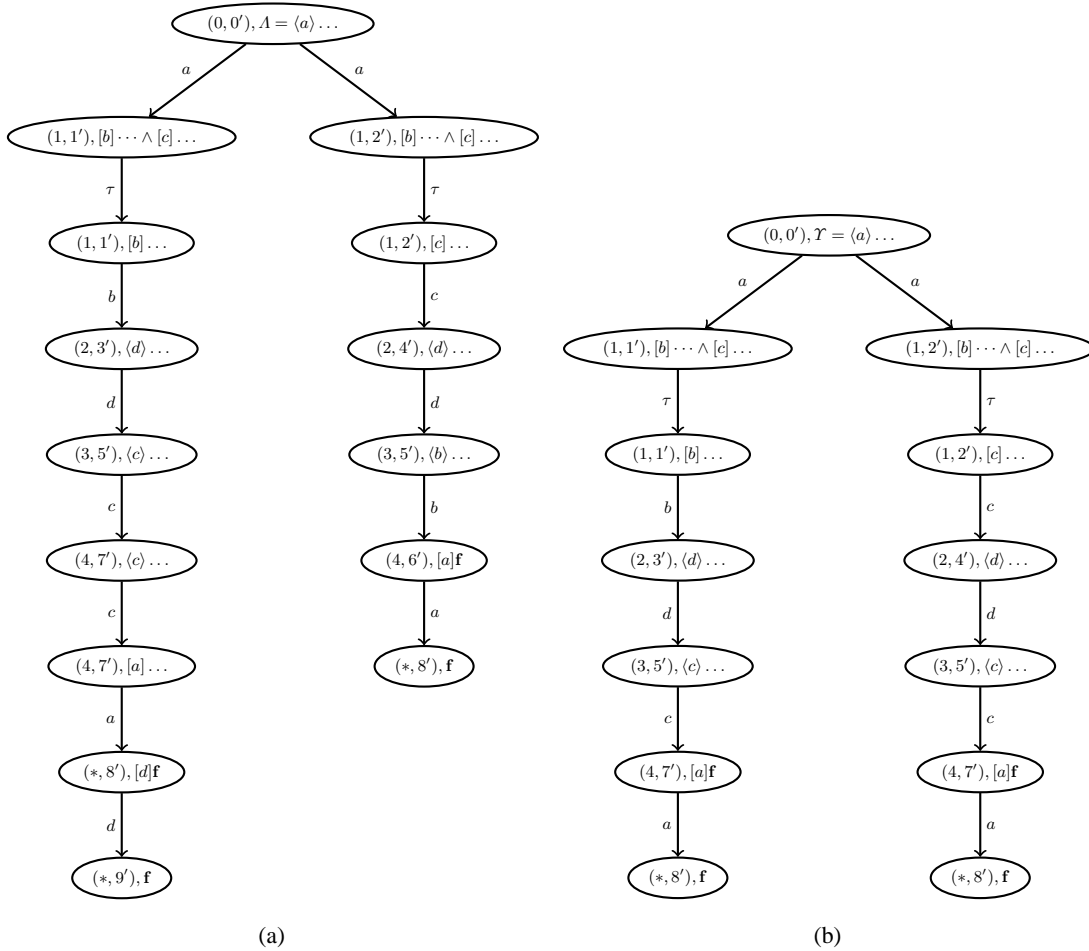
Fig. 16 (a) Combination of the proof-trees of Figure 15. τ transitions between identical nodes have been ignored to reduce the size of the tree. (b) Reduction of the tree in part (a).

violating condition (5) of Definition 16, and there is a transition stemming from a disagreement state $(*, 8')$ on a which violates condition (3) of Definition 16.

A.2 Reducing the combination

Now we prune the tree, removing parts of the proof which are not needed to show the explanation of the inconsistency. We begin by removing loops. Suppose that two states are inconsistent and there are two ways of showing it, in terms of proofs of formulas φ and φ'. If φ' is a subformula of φ, the witness φ' is smaller and therefore would be preferred. Similarly, the witness is obtained as long as one of the proof-trees being combined has reached a leaf. More formally, we apply the following procedure to $\Pi^{\varphi}_{(M,N)}$:
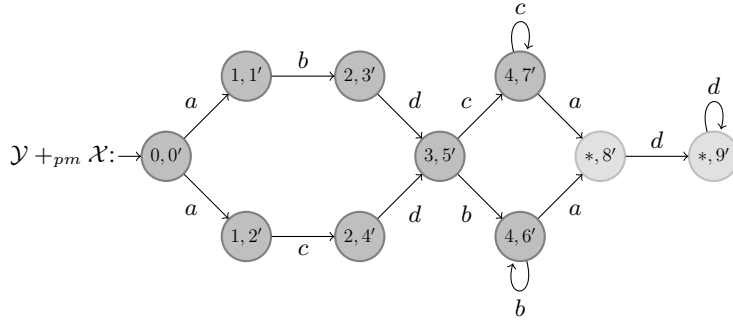
Fig. 17 Pseudo-merge of the models of Figure 14.

Step 1 Replace each non-leaf node $((m, *), \varphi')$ by a leaf $((m, *), \mathbf{t})$, and each non-leaf node $((*, n), \varphi')$ by a leaf $((*, n), \mathbf{f})$.

Step 2 For each node pair (m, n), find the set $F_{m,n}$ of all nodes of the form $((m, n), \varphi'')$. Aiming to produce the shortest proof (and thus the smallest distinguishing formula), we choose those that cannot be further reduced. We do this by selecting a node $((m, n), \varphi'') \in F_{m,n}$ such that no descendent of it is in $F_{m,n}$, replacing all subtrees rooted in $((m, n), \varphi')$ by the one rooted in $((m, n), \varphi'')$.

Step 3 For each node $((m, n), [\ell]\varphi')$ with only one child $((m', n'), \varphi'')$, if $m \xrightarrow{\ell}_r m' \in M$, then change $[\ell]\varphi'$ into $\langle \ell \rangle \varphi'$. This is necessary in cases where a path is an explanation for a formula and its negation. The $\mathcal{F}$ operator arbitrarily chooses the formula to be true in the first model and false in the other; this step anticipates this choice when building the new formula.

Step 4 Propagate the changes made to the formulas bottom-up. Since we have kept track of the correspondences between children and subformulas, if the child has changed, we update the corresponding subformula in the parent. Since we have made copies of the formulas, a change in one subtree does not affect its sibling.

The result of this procedure is a new tree $\Pi^{\varphi'}_{(M,N)}$. Note that all τ transitions have been collapsed in step 2. Therefore, all formulas are either constants $\mathbf{t}$ and $\mathbf{f}$, or start with $\langle \ell \rangle$ or $[\ell]$. In the latter case, the outgoing transitions are labelled with ℓ.

In Section A.4, we show that this tree encodes the proofs of φ' in the original models M and N, that is, we can extract proof-trees $\Pi^{\varphi'}_M$ and $\Pi^{\neg\varphi'}_N$ from it.

Example Applying the procedure described above to our example means replacing the branch

$$(4, 7'), [a][d]\mathbf{f} \xrightarrow{a} (*, 8'), [d]\mathbf{f} \xrightarrow{d} (*, 9'), \mathbf{f}$$

by

$$(4, 7'), [a]\mathbf{f} \xrightarrow{a} (*, 8'), \mathbf{f}.$$

Then we note that the pair of states $(3, 5')$ appears twice (we do not count the trivial collapsing of τ transitions). Since neither of the $(3, 5')$ nodes are ancestors of one another, we can choose either of them to replace the other. Suppose that we choose to keep the subtree for the formula $\langle c \rangle \langle c \rangle [a][a]\mathbf{f}$ (note that the changes made to one of the branches of the proof-tree are not yet reflected in the formula). This still gives us two occurrences of the node $(4, 7')$. In that case, the one labelled with $\langle c \rangle [a][a]\mathbf{f}$ is an ancestor of the one labelled with $[a][a]\mathbf{f}$. We therefore keep the latter and replace the former. Finally, we update the labels of the formulas, starting from the leaves, yielding the new tree of Figure 16(b). The transformation of the tree yields a transformation of the formulas, i.e., instead of the original formula Λ, the root of this tree is now labelled by the formula

$$\Upsilon = \langle a \rangle ([b]\langle d \rangle \langle c \rangle [a]\mathbf{f} \wedge [c]\langle d \rangle \langle c \rangle [a]\mathbf{f}).$$

Note that the graph we obtain by projecting this tree onto the pseudo-merge is a distinguishing DAG.

A.3 Obtaining a distinguishing DAG

Here we show that the distinguishing formula is induced by the pseudo-merge. Let us consider the projection of $\Pi^{\varphi'}_{(M,N)}$ onto $M +_{pm} N$. This projection is possible because each non-τ transition in the original tree $\Pi^{\varphi}_{(M,N)}$ corresponds to at least one required transition in an original model, thus complying with condition (2) of Definition 16. The reason for this is as follows. Rules in Figure 7 (except **MM**, $\mathbf{\Gamma}*$, or $*\mathbf{\Gamma}$ when $\gamma = m$) add a required transition to $M +_{pm} N$. This transition is not removed because removal happens only if there is no inconsistency on a label ℓ and we assume that a distinguishing property starts with ℓ. Since the transformation of $\Pi^{\varphi}_{(M,N)}$ into $\Pi^{\varphi'}_{(M,N)}$ collapses only those nodes that share the same labels for states, and never adds transitions, we can still project it onto the pseudo-merge.

Moreover, step 2 of the transformation removes all loops. Therefore, the projection is a DAG, as required by condition (1) of Definition 16. Step 1 has removes all transitions from nodes corresponding to $(m, *)$ or $(*, n)$. Hence. the projection satisfies condition (3) of Definition 16.

In addition, all leaves of the proof are either created by step 1 or are copies of leaves of the original proof, created either by case (iii) of Definition 19, followed by an application of case (i), or, dually, by case (iv) followed by case (i). In all such cases, the leaves are disagreement states, and condition (4) of Definition 16 holds.

By step 2, all nodes corresponding to a pair of states correspond to the same formula, with the outgoing transition from these labelled by the first letter of the formula. Thus, condition (5) holds.

Finally, in cases where the formula starts with $\langle \ell \rangle$, we have combined a proof-tree for $\langle \ell \rangle \varphi$ with one for $[\ell]\neg\varphi$. By construction, we consider a single required transition in M and all possible transitions in N. By a dual reasoning, condition (6) of Definition 16 holds in the case when the formula starts with $[\ell]$. Therefore, we can project $\Pi^{\varphi'}_{(M,N)}$ onto the pseudo-merge and obtain a distinguishing DAG.

A.4 Extracting a formula

We now show that φ' is exactly the formula obtained by applying the $\mathcal{F}$ operator on the DAG G built in Section A.3. The proof is by structural induction on the size of G.

By step 1 of the transformation, cases (i) and (ii) of Definition 17 guarantee that $\mathcal{F}$ applied to a DAG with a single transition yields a formula that labels the corresponding proof-tree.

Inductive hypothesis: Suppose that $\mathcal{F}$ applied to any subgraph of G yields the formula that labels the corresponding proof-tree.

Inductive case: Suppose case (iii) of the $\mathcal{F}$ operator is applied. That means that we have used the case (v) of the $Comb$ operator to produce the tree (or the subtree that has now replaced it). In that case, we have introduced a conjunction of formulas, one for each child. Even if the formulas were originally identical, they may have changed during the transformation. In all cases, the propagation of changes by step 4 changed the placeholder into the formula that now labels the corresponding child. Since each child is labelled with the formula that could have been produced by the $\mathcal{F}$ operator over the projection, so does the parent. Case (iv) being dual, nodes are always labelled with the formula that would have been produced by the application of the $\mathcal{F}$ operator over the projection of $\Pi^{\varphi'}_{(M,N)}$ over $M +_{pm} N$.

Therefore, applying $\mathcal{F}$ to the initial state of the DAG yields the formula φ'.

A.5 Building new proof-trees

We now show that φ' produced in Section A.4 is still distinguishing by producing proof-trees for each model. We do so by applying the *Split* operator defined below to the tree $\Pi^{\varphi'}_{(M,N)}$, resulting in proof-trees $\Pi^{\varphi'}_M$ and $\Pi^{\neg\varphi'}_N$.

Definition 20 (The *Split* operator) *Let T be a tree whose nodes are in $(S_1 \times S_2) \times \mathcal{L}^p_\mu$. The* Split *operator is defined inductively as follows, with $\bot$ being the empty tree; transitions leading to $\bot$ are ignored:*

(i) $Split((m, *), \mathbf{t}) = ((m, \mathbf{t}), \bot)$

(ii) $Split((*, n), \mathbf{f}) = (\bot, (n, \mathbf{t}))$

(iii) $Split(((m, n), \langle \ell \rangle \bigwedge_{i=1}^k \varphi_i) \xrightarrow{\ell} \{((m', n'_i), \varphi_i)\}_i) =$
$\quad ((m, \langle \ell \rangle \bigwedge_{i=1}^k \varphi_i) \xrightarrow{\ell} (m', \bigwedge_{i=1}^k \varphi_i) \xrightarrow{\tau} \{\Pi^{\varphi_i}_{m'}\}_i, (n, [\ell] \bigvee_{i=1}^k \neg\varphi_i) \xrightarrow{\ell} \{(n'_i, \bigvee_{i=1}^k \neg\varphi_i) \xrightarrow{\tau} \Pi^{\neg\varphi_i}_{n'_i}\}_i)$
$\quad$ *where* $\forall i \in \{1, \ldots, k\}, (\Pi^{\varphi_i}_{m'}, \Pi^{\neg\varphi_i}_{n'_i}) = Split(((m', n'_i), \varphi_i))$

(iv) $Split(((m, n), [\ell] \bigvee_{i=1}^k \varphi_i) \xrightarrow{\ell} \{((m'_i, n'), \varphi_i)\}_i) =$
$\quad ((m, [\ell] \bigvee_{i=1}^k \varphi_i) \xrightarrow{\ell} \{(m'_i, \bigvee_{i=1}^k \varphi_i) \xrightarrow{\tau} \Pi^{\varphi_i}_{m'_i}\}_i, (n, \langle \ell \rangle \bigwedge_{i=1}^k \neg\varphi_i) \xrightarrow{\ell} (n', \bigwedge_{i=1}^k \neg\varphi_i) \xrightarrow{\tau} \{\Pi^{\varphi_i}_{n'}\}_i)$
$\quad$ *where* $\forall i \in \{1, \ldots, k\}, (\Pi^{\varphi_i}_{m'_i}, \Pi^{\neg\varphi_i}_{n'}) = Split(((m'_i, n'), \varphi_i))$

If none of the above cases is applicable, then the result of the $Split$ operator is undefined.
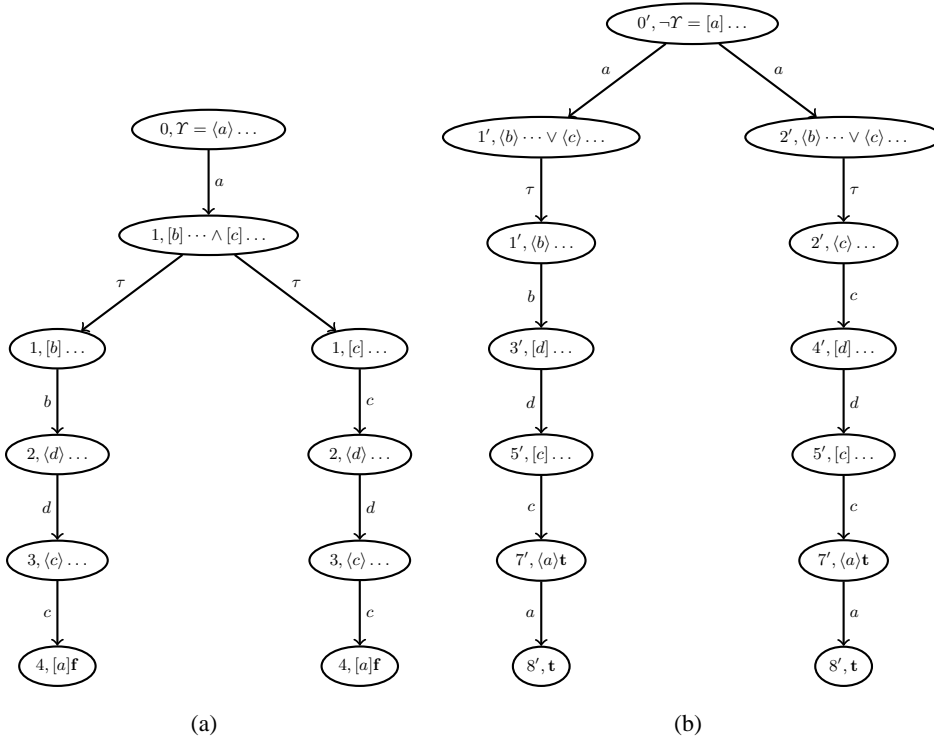
Fig. 18 The proof-trees obtained from the reduced tree of Figure 16(b): (a) $\mathcal{X} \vDash \Upsilon$ and (b) $\mathcal{Y} \vDash \neg \Upsilon$.

Note that because of the form of trees built by our algorithms, the above definition is total. That is, one of the cases (i)-(iv) always applies. To prove that the trees produced by *Split* are indeed proof-trees, we just have to show that in the case of $[\ell]$ (resp. $\langle \ell \rangle$), we consider all possible transitions on ℓ in M (resp. N). This is the case because we started with proof-trees and never removed just transitions; instead, we replaced nodes by other nodes from the tree. When we did remove transitions, we removed all of them, changing the formula accordingly.

Finally, we show that the proofs we produced are subgraphs of the ones we initially had. Since we never added transitions and only replaced nodes with those with the same labels, all transitions of the projection of $\Pi_M^{\varphi'}$ over M (resp. $\Pi_N^{\neg\varphi'}$ over N) are also transitions of the projection of Π_M^{φ} over M (resp. $\Pi_N^{\neg\varphi}$ over N).

Example   When we split the tree of Figure 16(b), we obtain the proof-trees of Figure 18. The projection of each of these new proof-trees is a subgraph of the projection of the original corresponding proof-tree, as expected.

B Algorithm to build a distinguishing DAG

In this section, we describe the algorithm BUILDDAG. It builds a path to the targeted boundary disagreement transition. The algorithm uses the $\sqcup$ operator, defined below, that computes a "union" of sets whose elements are of

the form (a, B), where B is a set. The elements having the same head (e.g., (a, B) and (a, C)) are merged into a single pair by merging B and C. For example,

$$\{(a, \{1, 2\}), (b, \{2, 3\}), (c, \{1, 3\})\} \sqcup \{(a, \{2, 3\}), (c, \{0, 1\})\} = \{(a, \{1, 2, 3\}), (b, \{2, 3\}), (c, \{0, 1, 3\})\}$$

Definition 21 ($S \sqcup T$) *Let S and T are sets of pairs of the form (a, B), where B is a set. Then,*

$$S \sqcup T = \{(a, B \cup C) | (a, B) \in S \wedge (a, C) \in T\} \cup \{(a, B) | (a, B) \in S \wedge \nexists C \cdot (a, C) \in T\} \cup$$
$$\{(a, C) | (a, C) \in T \wedge \nexists B \cdot (a, B) \in S\}$$

The algorithm BUILDDAG uses a sub-routine to add the needed transitions to comply with the conditions required by the definition of a distinguishing DAG (Definition 16). The newly added transitions may require additional transitions in order to actually obtain a distinguishing DAG, hence the DAG is computed as a fixpoint.

Recall the example in Figure 11 of Section 6. It showed that there are cases when M and N are inconsistent MTSs over the same alphabet and yet a distinguishing DAG containing a given (boundary disagreement) transition does not exist. We thus define the following notion of correctness of Algorithm 1:

Property 3 (Correctness of Algorithm 1) Let $M = \langle S_M, Act, \Delta_M^r, \Delta_M^p, m_0 \rangle$ and $N = \langle S_N, Act, \Delta_N^r, \Delta_N^p, n_0 \rangle$ be inconsistent MTSs. If the distinguishing DAG of $P = M +_{pm} N$ with the user-specified transition $(m_u, n_u) \overset{\ell}{\longrightarrow}_r$ $(m'_u, n'_u) \in P$ exists, Algorithm 1 computes it.

Proof We shall show that the graph that is built by BUILDDAG is indeed a distinguishing DAG, as defined by Definition 16. First, we do build a DAG, because before adding new vertices, we check that the pair of states is not in the set F of ancestors in the graph we are constructing, preventing the inclusion of any cycle in the graph. In each case of the call to SUBROUTINE, every transition we add corresponds to a required transition at the leaf of one of the models. Therefore, by one of the rules **RR**, **RM**, **MR**, **RF**, **FR**, the transition we are adding to the DAG should be a required one. Because we never add states from $D_M \cup D_N$ in sets I or J, we never take transitions that stem from one of these in the DAG. On the other hand, a transition is a leaf only if it has been built by one of the two first cases of SUBROUTINE: otherwise, we add its targets to one of the set I or J, unless it has already been visited. In both cases, the pair has been or will be visited, and visiting a state means adding transition from it in the DAG. As we visit a given pair of states only once, and since an application of the SUBROUTINE adds transitions labelled by only one letter, all transitions stemming from a node of our DAG will have the same letter. Finally, the conditions of lines 10 and 19 of the algorithm ensure that we add all transitions on a given symbol for one of the MTSs and only a single, required, transition for the other. Therefore, the graph returned by BUILDDAG is a distinguishing DAG. $\square$

Algorithm 1 BuildDAG

1: **procedure** BUILDDAG(M, N)

2:     Let $\langle S_M, Act, \Delta_M^{\mathsf{r}}, \Delta_M^{\mathsf{p}}, m_0 \rangle = M$

3:     Let $\langle S_N, Act, \Delta_N^{\mathsf{r}}, \Delta_N^{\mathsf{p}}, n_0 \rangle = N$

4:     Let $P = M +_{pm} N$

5:     Choose $(m_u, n_u) \xrightarrow{\ell}_{\mathsf{r}} (m'_u, n'_u) \in P$ such that $(m'_u, n'_u) \in D_M \cup D_N$ ($*$ Chosen by the user $*$)

6:     Let π be a *required agreement* path with no loop to (m_1, n_1) traversing only pairs of inconsistent states

7:     $\pi = \pi :: ((m_u, n_u) \xrightarrow{\ell}_{\mathsf{r}} (m'_u, n'_u))$ ($*$ Extend π with our goal transition $*$)

8:     $X = \emptyset$ ($*$ Set of visited states $*$)

9:     Let $I = \{((m, n), \{(m, n)\}) | (m, n) \in \pi\}$ ($*$ Pairs of states to traverse first $*$)

10:    Let $J = \emptyset$ ($*$ Pairs of states to traverse $*$)

11:    Let $G = ((m_0, n_0), V, \Delta_G) = ((m_0, n_0), \{(m_0, n_0)\}, \emptyset)$ ($*$ A placeholder for the DAG $*$)

12:    **while** $I \cup J \neq \emptyset$ **do**

13:        **if** $I = \emptyset$ **then**

14:            Let $(m, n), F = J.\texttt{pop()}$

15:        **else**

16:            Let $(m, n), F = I.\texttt{pop()}$

17:        $X = X \cup \{(m, n)\}$

18:        **if** $(m, n) \xrightarrow{a}_{\mathsf{r}} (m', n')$ is a step in π **then**

                                                                                 ($*$ Try to apply the normal algorithm with letter a $*$)

19:            Let success = SUBROUTINE($m, n, F, \{m'\}, \{n'\}, \{a\}$)

20:            **if** $\neg$ success **then**

21:                Backtrack

22:        **else**

23:            Let success = SUBROUTINE(m, n, F, S_M, S_N, Act)   ($*$ Apply the normal algorithm $*$)

24:            **if** $\neg$ success **then**

25:                Backtrack

26:    **return** G

Algorithm 2 SubRoutine

1: **procedure** SUBROUTINE($m, n, F, targets_M, targets_N, subAct$)

2:     **if** $m \xrightarrow{a}_r m' \land n \not\xrightarrow{a} \land m' \in targets_M \land a \in subAct$ **then**   ($*$ Rule **RF** $*$)

3:         $V = V \cup \{(m', *)\}$

4:         $\Delta_G = \Delta_G \cup (m, n) \xrightarrow{a}_r (m', *)$

5:         **return** True

6:     **else if** $n \xrightarrow{a}_r n' \land m \not\xrightarrow{a} \land n' \in targets_N \land a \in subAct$ **then**   ($*$ Rule **FR** $*$)

7:         $V = V \cup \{(*, n)\}$

8:         $\Delta_G = \Delta_G \cup (m, n) \xrightarrow{a}_r (*, n')$

9:         **return** True

10:     **else if** $m \xrightarrow{a}_r m' \land (\forall n', n \xrightarrow{a}_p n' \Rightarrow \neg\mathrm{Cons}(m', n') \land (m', n') \notin F) \land m' \in targets_M \land a \in subAct$ **then**
         ($*$ Rule **RR** or **RM** $*$)

11:         Let $T = \{n' | n \xrightarrow{a}_p n'\}$

12:         $V = V \cup \{(m', n') | n' \in T\}$

13:         $\Delta_G = \Delta_G \cup \{(m, n) \xrightarrow{a}_r (m', n') | n' \in T\}$

14:         **for** $n' \in T$ such that $(m, n) \xrightarrow{a}_r (m', n') \in \pi$ **do**

15:             $I = (I \sqcup \{((m', n'), F \cup \{(m', n')\})\}) \setminus X$

16:             $T.\texttt{remove}(n')$

17:         $J = (J \sqcup \{((m', n'), F \cup \{(m', n')\}) | n' \in T\}) \setminus X$

18:         **return** True

19:     **else if** $n \xrightarrow{a}_r n' \land (\forall m', m \xrightarrow{a}_p m' \Rightarrow \neg\mathrm{Cons}(m', n') \land (m', n') \notin F) \land n' \in targets_N \land a \in subAct$ **then**
         ($*$ Rule **RR** or **MR** $*$)

20:         Let $T = \{m' | m \xrightarrow{a}_p m'\}$

21:         $V = V \cup \{(m', n') | n' \in T\}$

22:         $\Delta_G = \Delta_G \cup \{(m, n) \xrightarrow{a}_r (m', n') | n' \in T\}$

23:         **for** $m' \in T$ such that $(m, n) \xrightarrow{a}_r (m', n') \in \pi$ **do**

24:             $I = (I \sqcup \{((m', n'), F \cup \{(m', n')\})\}) \setminus X$

25:             $T.\texttt{remove}(m')$

26:         $J = (J \sqcup \{((m', n'), F \cup \{(m', n')\}) | m' \in T\}) \setminus X$

27:         **return** True

28:     **else**

29:         **return** False