Exploring Inconsistencies between Modal Transition Systems

Mathieu Sassolas
mathieu.sassolas@ens-cachan.fr

MPRI 2nd year internship with Sebastián Uchitel at the *Universidad de Buenos Aires*

March - July 2008

Abstract

In this document I present the work done under the supervision of Sebastian Uchitel between March and July 2008, in the research team of the *Laboratorio de Fundamentos y Herramientas para la Ingenieriá de Software* (LaFHIS) at the *Universidad de Buenos Aires* (UBA).
In this work we define a formalism to represent inconsistencies between two Modal Transition Systems that cannot be merged in an augmented model we call a pseudo-merge. We present a way to build such a model that encodes all shortest explanations of inconsistencies. These explanations are in the form of a distinguishing μ-calculus property. We also show how to represent graphically these formal explanations and how we intend the user to guide the generation of feedback.

Aknowledgements

I would like to thank specially Sebastián Uchitel for welcoming me in his team in Buenos Aires, for his time and advice all through the length of my internship. I would also like to thank Marsha Chechik of the *University of Toronto* for her questions and advice. Finally, I would like to thank all the people of the LaFHIS (Fernando Asteasuain, Victor Braberman, Guido de Caso, Hernán Czemerinski, Nicolás D'Ippolito, Diego Garbervetsky, Hernán Melgratti, Esteban Pavese, Fernando Schapachnik, Marité Segui) and more generally all the researchers of the department of computer science (departamento de computación) of the *Universidad de Buenos Aires* for their warm welcome.

Contents

1 Introduction **3**

2 Motivation **4**

3 Background **6**
3.1 Transitions systems 6
3.2 The underlaying logic: μ-calculus 6
3.3 Consistency, refinement, and merge 7

4 Explaining Inconsistency Graphically **8**

5 Characterization of Inconsistency **9**
5.1 Pseudo-merge 10
5.2 Correctness and Completeness 12

6 User-guided Feedback Generation **14**

7 Validation **18**
7.1 The special case of LTSs 18
7.2 The printer case study 18
7.3 The safety injection system 19

8 Conclusion and future work **21**

A Proof of selected properties **22**
A.1 Proof of completeness of the pseudo-merge 22
A.1.1 Combining the proof-trees 22
A.1.2 Reducing the combination 24
A.1.3 Having a distinguishing DAG 25
A.1.4 Extracting a formula 26
A.1.5 Building new proof-trees 26

List of Figures

1 Two systems, $\mathcal{A}$ and $\mathcal{B}$, of the law-making process. 4
2 Graphical feedback on the law-making models 5
3 Pseudo-merge of models $\mathcal{A}$ and $\mathcal{B}$ 6
4 Rules for the $+_l$ operator. 8
5 Proof trees 9
6 Rules for the pseudo-merge operator. 11
7 Definition of transitions for $M +_{pm} N$. 12

8 Formalization of condition (6) of Definition 14. 12
9 A case when we cannot build a distinguishing DAG containing a given transition 18
10 Models of a printer 19
11 The pseudo-merge of the printer models 20
12 Models used to illustrate completeness of the pseudo-merge. 22
13 Original proof-trees used to illustrate completeness 23
14 Combination of the proof-trees of Figure 13 and a reduction of it. 24
15 Pseudo-merge of the models of Figure 12 25
16 The proof-trees obtained from the reduced tree of Figure 14(b) 27

List of Algorithms

1 BuildDistinguishingDAG 15
2 SubRoutine for BuildDistinguishingDAG 16

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 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 [5, 10] provide a framework for comparison that abstracts away syntactic differences in behaviour descriptions. Refinement notions such as those based on simulation [11] support checking whether one model has been further elaborated than another. In addition, merging [13] allows combining two partial consistent descriptions into a comprehensive description that is a refinement of the models being merged.

Notions of equivalence, refinement and merge are crucial for behaviour model elaboration, and tools that support behaviour modelling with variants of these analyses have been developed, e.g., [1]. Yet little support is provided by existing approaches and tools to understand why two models are inconsistent and hence cannot be merged, or why one model is not a refinement of another.

Model-checkers provide useful feedback in the form of traces representing "executions" of the models. Such feedback can be computed for violations of behavioural properties. Yet 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 [3], 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 specific context of Modal Transition Systems (MTSs). The MTS formalism is an extension of traditional Labelled Transition Systems (LTSs) that supports operational descriptions of system behaviour for which certain aspects of behaviour are explicitly represented as *unknowns*. MTSs naturally support conjunction of partial knowledge of system behaviour through the notion of merge [13]. That is, an MTS which composes the information of two *consistent* partial models can be constructed through a merge operation. However, if the MTSs to be merged are inconsistent, we would like to help users understand sources of such inconsistencies and potentially fix them.

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

A special case of the results presented in this paper is the exploration of inconsistencies in traditional modelling techniques such as LTS as opposed to the more general setting of partial behaviour modelling in which we present our results. For example, given that bisimulation of LTSs is a special case of refinement of MTSs, it is possible to use the approach to provide feedback on the differences between two non-equivalence of 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 merging process in general. In Section 4

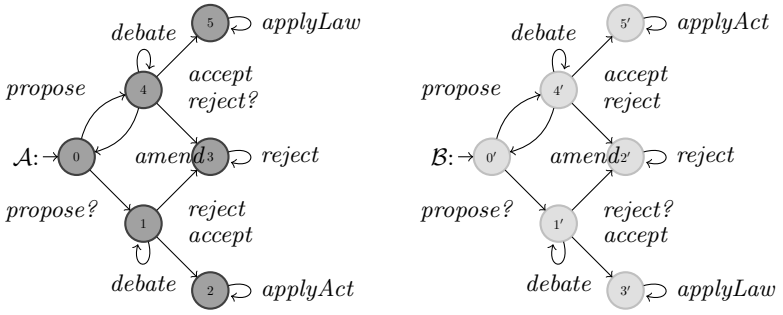


Figure 1: Two systems, $\mathcal{A}$ and $\mathcal{B}$, of the law-making process.

we present 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 inconsistent MTSs and use it to compute feedback on the cause of inconsistency in the form of a μ-calculus formula. In Section 6, we present a method for the user to guide the generation of this feedback. We present applications of this technique to help reason about bisimulation failure between LTSs and other case studies in Section 7 . Finally, Section 8 presents conclusions and discusses future work.

2 Motivation

In this section, we discuss two different behaviour models that describe the process for passing laws in a fictitious assembly. More specifically, we discuss the feedback that can be provided to explain their differences.

Consider models in Figure 1 (for the purpose of this section, ignore the "?" symbols that appear on some labels). They describe a process in which texts are produced and, after some debate, are 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 (see *propose* transitions from states 0 and $0'$). Since laws and acts are to be applied differently, the protocol for passing them differs well. The models in Figure 1 differ in which 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 [2]. In this case, to show the user the *cause* for inconsistency, it would be appropriate to produce a trace that is possible in one of the models but is forbidden in the other. However, traces may not be sufficient: for example, the above models disagree on when amendments can occur but have the same traces! Consider the trace *propose*, *amend*, *propose*, *accept*, *applyLaw*, ... which can be exhibited by both models. We know that in the first model, the original proposal must have corresponded to a Law that was later amended, while in the second model, the original proposal must have been an Act. In one of the models, the state reached after *propose* has the potential for accepting an *applyLaw* without amendments, while the second has a similar potential for Acts. As this example shows, the problem of using traces on their own as feedback is that they lack information on the (branching) structure of the two models.

An alternative, more appropriate form of feedback is to provide a *tree structure* to distinguish between the two models. For instance, consider the tree in Figure 2(a). It conveys that the potential behaviour after *propose* is to *amend*, or to *accept* and *applyLaw*. This tree is computed using states and transitions of the first model and is shown graphically by overlaying the tree over the structure of the model (see the dashed transitions in Figure 2(b)). However, a corresponding tree cannot be produced using the second model (see Figure 2(c) where dotted transitions are an explicit representation of what cannot be done): if the *propose* transition to state $4'$ is taken, then *amend* is possible but *accept*, *applyLaw* is not; on the other hand, if the *propose* transition to $1'$ is taken, then *accept*, *applyLaw* is possible but *amend* is not. In this paper (see Section 6), we show how to produce such graphical feedback automatically.

Formally, the tree in Figure 2(a) characterizes a modal μ-calculus ($\mathcal{L}_\mu$) formula that distinguishes

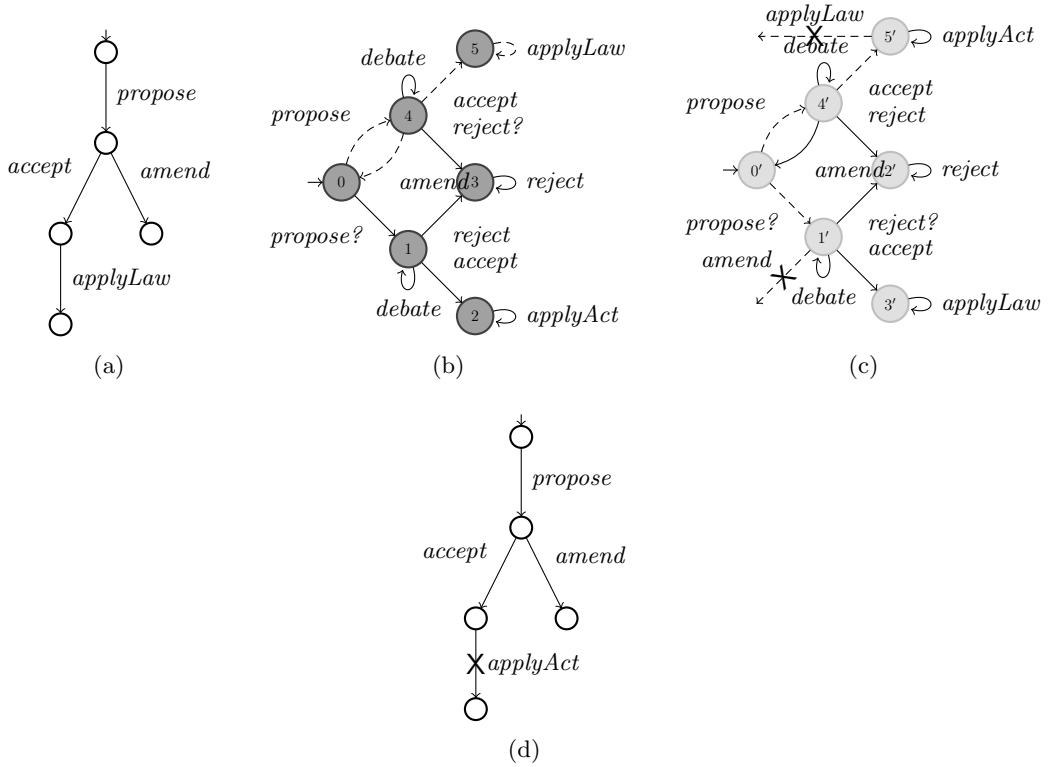


Figure 2: Graphical feedback for a formula ψ. (a) A simplified proof-tree;(b) and (c) Projections of (a) onto models $\mathcal{A}$ and $\mathcal{B}$;(d) Another simplified proof-tree.

between the two models; in other words, that evaluates to *true* in the first model and to *false* in the second. For the law-making example, such a formula can be

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

which states that there is a *propose* transition leading to a state that can both display the *accept*, *applyLaw*, and the *amend* behaviours. Furthermore, the projection of this tree onto one of the models (Figure 2(b)) represents a *proof* that the property holds, while the projection onto the other (Figure 2(c)), represents a *counterexample* to the formula. We give a more formal and complete definition of these trees in Section 4. We show the soundness of the graphical feedback produced by our approach in Section 5.

The tree depicted in Figure 2(a) can be interpreted intuitively as stating that the models disagree as to whether texts with *the potential to be passed as laws* can be amended. However, this is not the only way in which the difference between these two models can be explained. An explanation based on *the potential to amend acts* is also plausible and corresponds to the tree of Figure 2(d), yielding a different cause of inconsistency between the models being compared.

Thus, while individual trees suffice to explain the difference between two models, there are a number of them that can be proposed, each with a different relevance to the modellers. In our trivial example, the first tree 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 the second tree can prompt a discussion on whether acts-to-be may be amended leading to removal of a transition from $4'$ to $0'$ or addition of an *amend* transition from 1 to 0.

Although some explanation for inconsistency can be generated automatically, we support user guided generation of explanations by encoding all sources of inconsistency compactly and allowing exploration of this encoding. Figure 3 depicts a composition of the two law-passing models, $\mathcal{A}$ and $\mathcal{B}$, where highlighted states are the inconsistency points. 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 an inconsistency point (see Section 6). Selecting transition from $(5, 5')$ to $(5, *')$ generates the explanation shown in Figures 2(a) and 2(b)–(c), while selecting $(5, 5')$ to $(*, 5')$ yields the explanation in Figure 2(d).

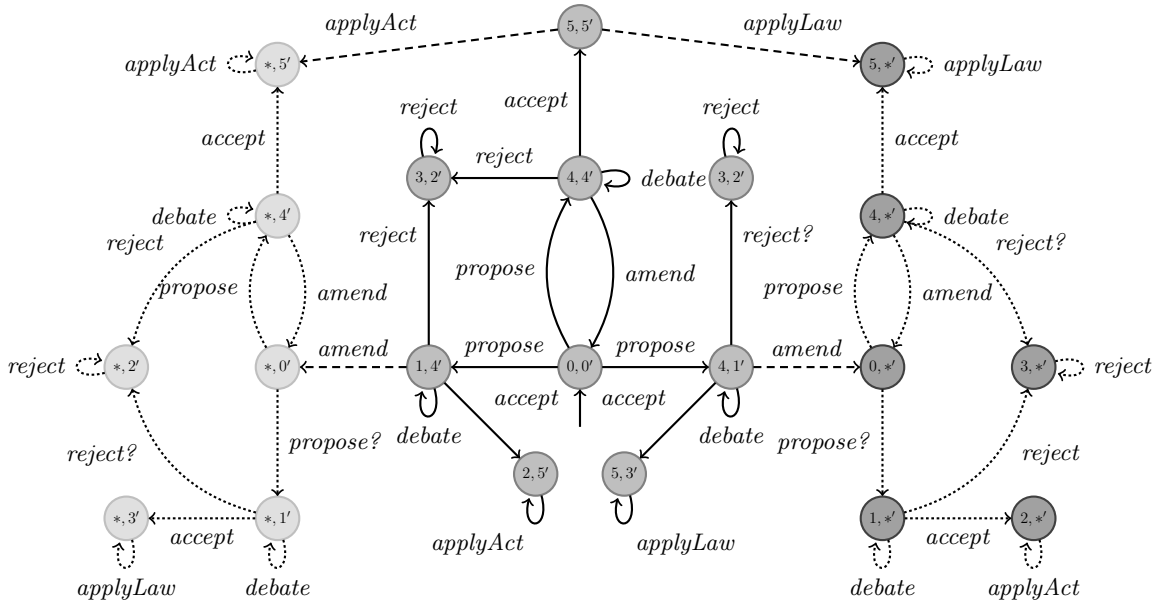


Figure 3: Pseudo-merge the law-making systems of Figure 1. State $3, 2'$ appears twice for clarity.

3 Background

In this section we present some notions and definitions we need in the rest of this report.

3.1 Transitions systems

The models we are going to reason on are Modal Transition Systems [8], a generalization of the well known Labelled Transition Systems [7].

Definition 1 (Labelled Transition System)**.** *A* labelled transition system *(LTS) is a tuple* $\langle S, Act, \Delta, s\rangle$ *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* $s \in S$ *the initial state.*

Definition 2 (Modal Transition System)**.** *A* modal transition system *(MTS) is a tuple* $\langle S, Act, \Delta^r, \Delta^p, s\rangle$ *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* $s \in S$ *is the initial state.*

We shall write $m \xrightarrow{\ell}_p m'$ when $(m, \ell, m') \in \Delta^p$, $m \xrightarrow{\ell}_r m'$ when $(m, \ell, m') \in \Delta^r$, $m \xrightarrow{\ell}_m m'$ when $(m, \ell, m') \in \Delta^p \setminus \Delta^r$, and $m \overset{\ell}{\not\longrightarrow}$ when $\forall m' \in S, (m, \ell, m') \notin \Delta^p$. Remark that an MTS in which $\Delta^r = \Delta^p$ is an LTS. For any state of an MTS, we can define a new MTS by just changing the initial state (and maybe removing some unreachable states). Hence we will often mix up a state of the MTS with this new MTS. We can also define another MTS by removing states.

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

3.2 The underlaying logic: μ-calculus

In order to reason over finite behaviors of MTSs, we use a subset of the modal 3-valued μ-calculus of [6] that does not include fixpoints. It is 3-valued to express the lack of knowledge over some transitions.

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

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

It is possible to remove all occurences of the negation according to the following rules:

(i) $\neg\mathbf{t} = \mathbf{f}$

(ii) $\neg\mathbf{f} = \mathbf{t}$

(iii) $\neg\langle\ell\rangle\varphi = [\ell]\neg\varphi$

(iv) $\neg[\ell]\varphi = \langle\ell\rangle\neg\varphi$

(v) $\neg(\varphi_1 \wedge \varphi_2) = \neg\varphi_1 \vee \neg\varphi_2$

(vi) $\neg(\varphi_1 \vee \varphi_2) = \neg\varphi_1 \wedge \neg\varphi_2$

The intuition of the $\langle\ell\rangle$ modularity is to say that there exist a required transition on ℓ satisfying the rest of the formula. Dually, the $[\ell]$ modularity expresses that all possible transitions on ℓ satisfy the rest of the formula. The semantics of the other operators is the usual one.

Definition 5 (Semantics of $\mathcal{L}^p_\mu$)**.** *For an MTS M and a $\mathcal{L}^p_\mu$ formula φ, φ is* true *in M, written $M \vDash \varphi$ according to the following rules:*

(i) $M \vDash \mathbf{t}$

(ii) $M \vDash \varphi_1 \wedge \varphi_2$ *iff* $M \vDash \varphi_1$ *and* $M \vDash \varphi_2$

(iii) $M \vDash \varphi_1 \vee \varphi_2$ *iff* $M \vDash \varphi_1$ *or* $M \vDash \varphi_2$

(iv) $M \vDash \langle\ell\rangle\varphi$ *iff* $\exists M' \cdot M \xrightarrow{\ell}_r M'$ *and* $M' \vDash \varphi$

(v) $M \vDash [\ell]\varphi$ *iff* $\forall M'$ *such that* $M \xrightarrow{\ell}_p M'$, $M' \vDash \varphi$

If $M \vDash \neg\varphi$, φ is false *in M. If it is neither the case that $M \vDash \varphi$ nor $M \vDash \neg\varphi$, then φ is* maybe *in M.*

To explore what differs between models, we use a special kind of properties, the ones that evaluate differently in the two models.

Definition 6 (Distinguishing property)**.** *For two MTSs M and N, a distinguishing property is a $\mathcal{L}^p_\mu$ formula φ such that $M \vDash \varphi$ and $N \vDash \neg\varphi$, or conversely.*

Note that if φ is distinguishing, then $\neg\varphi$ also is. In the following, we will often choose to always have a formula *true* in the first model and false in the second one, rather than the other way around.

3.3 Consistency, refinement, and merge

We need a notion of agreement (or rather absence of contradiction) between models in order to formally define the scope of our approach.

Definition 7 (Consistency relation)**.** *A* Consistency relation *between two MTSs M and N over the same alphabet Act is a binary relation $C_{M,N}$ such that $(M, N) \in C_{M,N}$ and the following conditions hold for all $\ell \in Act$ and all $(M', N') \in C_{M,N}$:*

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

If there exists a consistency relation between M and N, we say that the models (or states) are *consistent* and write it $\mathrm{Cons}(M, N)$. Otherwise, they are *inconsistent* and write $\neg\mathrm{Cons}(M, N)$.

Intuitively, refinement is the capacity for an MTS to add knowledge over previously unknown (*maybe*) behaviors, without changing any previously known behavior.

Definition 8 (Refinement)**.** *An MTS N is a* refinement *of M an MTS over the same alphabet, written $M \preceq N$ if there exists a refinement relation $\mathcal{R}$ such that:*

$$\frac{M \xrightarrow{\ell}_r M' \qquad N \xrightarrow{\ell}_r N'}{M +_l N \xrightarrow{\ell}_r M' +_l N'}\mathbf{TT} \qquad \frac{M \xrightarrow{\ell}_r M' \qquad N \xrightarrow{\ell}_m N'}{M +_l N \xrightarrow{\ell}_r M' +_l N'}\mathbf{TM}$$

$$\frac{M \xrightarrow{\ell}_m M' \qquad N \xrightarrow{\ell}_r N'}{M +_l N \xrightarrow{\ell}_r M' +_l N'}\mathbf{MT} \qquad \frac{M \xrightarrow{\ell}_m M' \qquad N \xrightarrow{\ell}_m N'}{M +_l N \xrightarrow{\ell}_m M' +_l N'}\mathbf{MM}$$

Figure 4: Rules for the $+_l$ operator.

1. $(M, N) \in \mathcal{R}$
2. $\forall (M', N') \in \mathcal{R},\ M' \xrightarrow{\ell}_r M'' \Rightarrow \left(\exists N'' \cdot N' \xrightarrow{\ell}_r N'' \wedge (M'', N'') \in \mathcal{R}\right)$
3. $\forall (M', N') \in \mathcal{R},\ N' \xrightarrow{\ell}_p N'' \Rightarrow \left(\exists M'' \cdot M' \xrightarrow{\ell}_p M'' \wedge (M'', N'') \in \mathcal{R}\right)$

One can refine a model until all behaviors are known, in which case we have an LTS, which we call an *implementation* of the MTS. Since different refinements will yield different LTSs, one can see an MTS as the (possibly infinite) set of its implementations.

The process of merging two MTSs M and N will try to agregate the knowledge contained in both of them into one model. Hence we will try to find a *minimal common refinement* of M and N. A common refinement will have all knowledge from M with some knowledge over some of its *maybe* behaviors, and conversely for N. We want minimality so that we do not add extra knowledge. In terms of sets of implementations, a merge will represent the intersection of the implementations of M and the implementations of N.

Finding a minimal common refinement is not always a simple problem [13, 4]. One obstacle that stops merging is inconsistency. Seen as sets of LTSs, two inconsistent MTSs have disjoint sets of implementations since they contain contradictory information. Therefore, inconsistent MTSs do not have any common refinement.

In terms of $\mathcal{L}^p_\mu$, refinement preserves *true* and *false* properties.

Property 1. *For M an MTS and φ an $\mathcal{L}^p_\mu$ formula, if $M \preceq N$ and $M \vDash \varphi$, then $N \vDash \varphi$.*

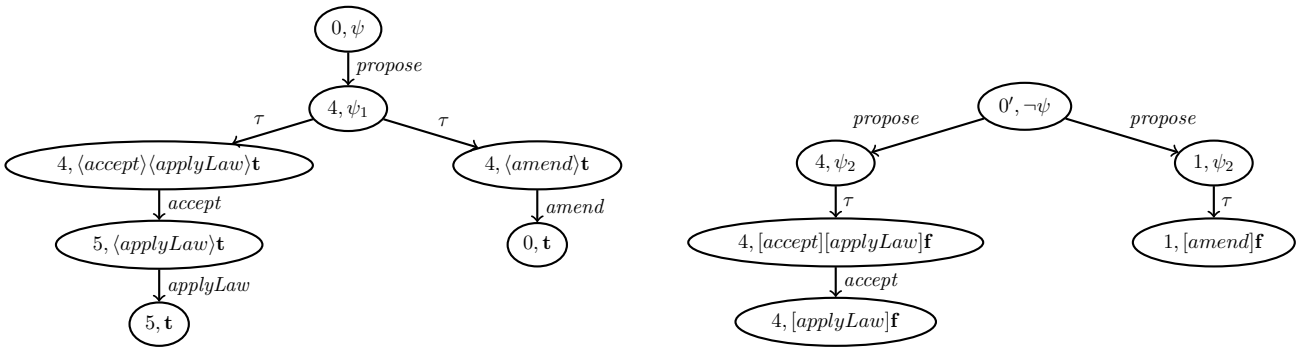
Note that the converse, that holds in the case of the whole μ-calculus [12], does not necessarily hold here. Therefore if there is a distinguishing property φ between M and N, there cannot be a common refinement, since φ would both be *true* and *false* in it. On the other hand, there is a common refinement if and only if the models are consistent, as shown in [4].

However, in the case in which the models are consistent, there is a way to build a common refinement (not alway minimal). Presented in [13], the $+_l$ operator considers only pairs of consistent states and applies the rules of Figure 4.

4 Explaining Inconsistency Graphically

In this section we explain how sound graphical feedback that explains why two models are inconsistent can be provided. In subsequent sections we show how such feedback can be automatically generated.

As shown in the previous section, the notion of inconsistency defined as the non-existence of a common refinement is characterised by the existence of a modal μ-calculus formula that evaluates to true in one of the models and false in the other. This means that μ-calculus formulas have sufficient expressiveness to provide feedback on inconsistency for all inconsistent models. As μ-calculus is clearly not an accessible language from a practitioners perspective, we show how graphical feedback in terms of two tree directed acyclic graphs – 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 formula does or does not hold in the inconsistent models.



(a) $\psi = \langle propose\rangle(\langle accept\rangle\langle applyLaw\rangle\mathbf{t} \wedge \langle amend\rangle\mathbf{t})$, $\psi_1 = \langle accept\rangle\langle applyLaw\rangle\mathbf{t} \wedge \langle amend\rangle\mathbf{t}$

(b) $\neg\psi = [propose]([accept][applyLaw]\mathbf{f} \vee [amend]\mathbf{f})$, $\psi_2 = \neg\psi_1 = [accept][applyLaw]\mathbf{f} \vee [amend]\mathbf{f}$

Figure 5: Proof trees for (a) formula ψ in model $\mathcal{A}$; (b) formula $\neg\psi$ in model $\mathcal{B}$.

We first define the notion of a proof tree. A *proof tree* for a model M and a modal μ-calculus formula ϕ encodes a proof that M satisfies ϕ. Each node of the tree consists of an MTS and a modal μ-calculus formula. Each transition is labelled with an action in $Act \cup \{\tau\}$. A node $n = \langle M, \phi\rangle$ has nodes $n_i = \langle M_i, \phi_1\rangle$ reached by transitions labelled ℓ_i only if $M \vDash \phi$ follows from $\forall i \cdot M_i \vDash \phi_i \wedge M \xrightarrow{\ell_i} M_i$. The proof tree for a model M and a formula ϕ has $\langle M, \phi\rangle$ as its root the node and all leaves have a formula which is either a tautology or a node of the form $\langle M, [\ell]\phi\rangle$ where there is no transition on ℓ from M.

Definition 9 (Proof Trees)**.** *Given a model M and property φ such that $M \vDash \varphi$, a proof tree for φ in M denoted $\Pi_{M,\varphi}$ of is a labelled tree $ProofTree(M, \varphi)$, inductively defined by:*

(i) $ProofTree(M, \mathbf{t}) = (M, \mathbf{t})$

(ii) $ProofTree(M, \langle\ell\rangle\varphi) = (M, \langle\ell\rangle\varphi) \xrightarrow{\ell} ProofTree(M', \varphi)$ *where* $M \xrightarrow{\ell}_r M'$ *and* $M' \vDash \varphi$.

(iii) $ProofTree(M, [\ell]\varphi) = (M, [\ell]\varphi) \xrightarrow{\ell} \{ProofTree(M'_1, \varphi), \ldots, ProofTree(M'_k, \varphi)\}$ *where the set* $\left\{M \xrightarrow{\ell}_p M'_i \middle| i \in \{1, \ldots, k\}\right\}$ *are all possible transitions on ℓ from M and* $\forall i \in \{1, \ldots, k\}, M'_i \vDash \varphi$.

(iv) $ProofTree(M, \bigwedge_{i=1}^{k} \varphi_i) = (M, \bigwedge_{i=1}^{k} \varphi_i) \xrightarrow{\tau} \{ProofTree(M, \varphi_1), \ldots, ProofTree(M, \varphi_k)\}$.

(v) $ProofTree(M, \bigvee_{i=1}^{k} \varphi_i) = (M, \bigvee_{i=1}^{k} \varphi_i) \xrightarrow{\tau} ProofTree(M, \varphi_j)$ *for some j such that* $M \vDash \varphi_j$ *and* $1 \leq j \leq k$.

Consider the models $\mathcal{A}$ and $\mathcal{B}$ in Figure 1 and the distinguishing property ψ in the caption which proves they are inconsistent. The proof that $\mathcal{A} \models \psi$ can be depicted in the proof tree shown in Figure 5(a). Note the conjunction in the formula is encoded as two separate branches.

An abstraction of a proof tree for a property φ on a model M can be depicted graphically over M by simply highlighting the portion of M that is covered by transitions in φ. For example consider Figure 2(b) which depicts using dashed lines the projection of the proof tree shown in Figure 5(a).

Definition 10 (Proof tree projection on a MTS)**.** *The projection N of a proof $\Pi_{M,\varphi}$ for φ in M is the subgraph of M such that* $m \xrightarrow{\ell} m' \in \rho \Leftrightarrow \exists \rho, \varphi' \cdot (m, \rho) \xrightarrow{\ell} (m', \varphi') \in \Pi_{M,\varphi}$.

To aid user comprehension, when projecting proof trees with leafs of the form $(m, [\ell]\phi)$, we explicitly display the ℓ transitions that are not possible in state M. For instance, the proof tree show in Figure 5(b) has a leaf labelled $(4, [applyLaw]\mathbf{f})$ which states that in state 4 of model B there are no *applyLaw* transitions. This fact is shown in the projection of the proof tree onto $\mathcal{B}$ in Figure 2(c), with a dashed crossed out transition.

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 merged: We define the set of formulas defined by a pseudo-merge and show that on one hand, all these properties are distinguishing (correctness), and on the other, that any distinguishing formula for the two models can be explained (via a shorter explanation) using pseudo-merge (completeness). 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 choose which of the many properties encoded in the pseudo-merge is to be used as feedback. The graphical representation of such properties has been discussed in Section 4.

5.1 Pseudo-merge

A pseudo-merge of MTSs M and N is an MTS where with two identified subsets of states: disagreement states for M (I_M) and disagreement states for N (I_N) such that if disagreement states for M (resp. N) are removed, then the pseudo-merge is a refinement of M (resp. 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. The transitions that go from agreement states to disagreement states 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 11 (Pseudo-merge)**.** *A* pseudo-merge *of two MTSs M and N with identical alphabets Act is a tuple $\langle P, I_M, I_N\rangle$, where P is an MTS $\langle S, Act, \Delta^r, \Delta^p, s\rangle$, $I_M \subseteq S$ and $I_N \subseteq S$ such that subMTSs of P over states $S \setminus I_M$ and $S \setminus I_N$ refine M and N, respectively. We call transitions in $S \times Act \times (I_M \cup I_N)$ and $(S \setminus (I_M \cup I_N)) \times Act \times (I_M \cup I_N)$* disagreement transitions *and* boundary disagreement transitions*, respectively.*

An example of a pseudo-merge for models $\mathcal{A}$ and $\mathcal{B}$ of Figures 1(a) and 1(b) is depicted in Figure 3. The disagreement states for $\mathcal{B}$ are labelled with a pair with $*$ as the first element and shown in light grey. The disagreement states for $\mathcal{A}$ are labelled with a pair with $*$ as the second element and shown dark grey. Boundary disagreement transitions are dashed, whereas the rest of the disagreement transitions are dotted.

We now present an algorithm, $+_{pm}$, for computing a pseudo-merge of two models. It an adaptation of the algorithm introduced in [13] for constructing common refinements of consistent MTSs. The algorithm, applied to models M and N, first builds a synchronous product by constructing the Cartesian product of $S_M \cup \{*\} \times S_N \cup \{*\}$, where S_M and S_N are the states of M and N and $*$ 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 synchronous product. Finally, the sets of disagreement states are defined as $I_N = (S_M \times \{*\})$ and $I_M = (\{*\} \times S_N)$. Intuitively, I_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 6. 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 TT. 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 has (N does not rule that the transition is required nor prohibited). 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 TM, while the symmetric situation is described in rule MT. 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

$$\frac{M \xrightarrow{\ell}_r M' \qquad N \xrightarrow{\ell}_r N'}{M +_{pm} N \xrightarrow{\ell}_r M' +_{pm} N'}\mathbf{TT} \qquad \frac{M \xrightarrow{\ell}_r M' \qquad N \xrightarrow{\ell}_m N'}{M +_{pm} N \xrightarrow{\ell}_r M' +_{pm} N'}\mathbf{TM} \qquad \frac{M \xrightarrow{\ell}_m M' \qquad N \xrightarrow{\ell}_r N'}{M +_{pm} N \xrightarrow{\ell}_r M' +_{pm} N'}\mathbf{MT}$$

$$\frac{M \xrightarrow{\ell}_m M' \qquad N \xrightarrow{\ell}_m N' \qquad \mathrm{Cons}(M', N')}{M +_{pm} N \xrightarrow{\ell}_m M' +_{pm} N'}\mathbf{MM} \qquad \frac{M \xrightarrow{\ell}_r M' \qquad N \not\xrightarrow{\ell}}{M +_{pm} N \xrightarrow{\ell}_r M' +_{pm} *}\mathbf{TF} \qquad \frac{M \not\xrightarrow{\ell} \qquad N \xrightarrow{\ell}_r N'}{M +_{pm} N \xrightarrow{\ell}_r * +_{pm} N'}\mathbf{FT}$$

$$\gamma \in \{t, m\}\frac{M \xrightarrow{\ell}_\gamma M'}{M +_{pm} N \xrightarrow{\ell}_\gamma M' +_{pm} *}\mathbf{\Gamma *} \qquad \gamma \in \{t, m\}\frac{N \xrightarrow{\ell}_\gamma N'}{* +_{pm} N \xrightarrow{\ell}_\gamma * +_{pm} N'}\mathbf{* \Gamma}$$

Figure 6: Rules for the pseudo-merge operator.

transit all together on ℓ. The rules also do not allow an ℓ transition on the synchronous product of M and N if they both agree in prohibiting ℓ transitions on M and N.

Rules FT and TF are of particular interest as they capture the situation in which M and N disagree. For instance, TF states that if an ℓ-transition is required in M but prohibited in N, then the synchronous product ℓ-transitions to a state which models that N has flagged the fact that a transition has occurred which is inconsistent with its own behaviour (the $*$ state). Rules $\Gamma*$ and $*\Gamma$ encode the synchronous product once one of the models has reached a has reached a $'*'$-state. Essentially, the synchronous product allows the model that is not in $*$ to transition freely while prohibiting any transitioning of the other model.

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 in M and N are consistent (recall Definition 7). 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 from the synchronous product 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) $m \xrightarrow{\ell}_r m'$ 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 $n \xrightarrow{\ell}_r n'$ occurs. The intuition for this rule is that M similarly to the 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 if this is the case then they do not represent proper disagreements between M and N. The rule states that if when computing the synchronous product, either of the models has a non-deterministic choice on ℓ, we wish to pair ℓ transitions, if possible, such that the resulting state in the synchronous product is consistent. In other words, we do not include pairings of ℓ transitions that lead to inconsistent states if there was a consistent option.

Definition 12 (The Synchronous Product)**.** *Let $M = (S_M, Act, \Delta^r_M, \Delta^p_M, s_{0_M})$ and $N = (S_N, Act, \Delta^r_N, \Delta^p_N, s_{0_N})$ be MTSs. Then a* synchronous product *of M and N is an MTS $P = \langle S_P, Act, \Delta^r_P, \Delta^p_P, p_0\rangle$, where $S_P = S_M \times S_N$, $p_0 = (s_{0_M}, s_{0_N})$, and Δ^r_P and Δ^p_P are the smallest relations that satisfy the rules given in Figure 6.*

Definition 13 (The $+_{pm}$ Operator)**.** *Let M, N and P be MTSs in Definition 12.* Result of $M +_{pm} N$ *is a tuple $\langle P', I_M, I_N\rangle$, where $I_N = (S_M \times \{*\})$, $I_M = (\{*\} \times S_N)$ and $P' = \langle S_P, Act, \Delta^r_{P'}, \Delta^p_{P'}, p_0\rangle$, where $\Delta^p_{P'}$ is defined as shown in Figure 7 and $\Delta^r_{P'} = \Delta^r_P \cap \Delta^p_{P'}$.*

Applying $+_{pm}$ to the models of the law-making process (see Figure 1), we obtain the model in Figure 3. 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

$$\Delta^p_{P'} = \Delta^p_P \setminus \{(m,n) \xrightarrow{\ell}_p (m',n') \in \Delta^p_P \mid \neg\mathrm{Cons}(m',n') \;\wedge\; (m \not\xrightarrow{\ell}_r m' \vee \exists n'' \cdot (\mathrm{Cons}(m',n'') \wedge n \xrightarrow{\ell}_p n'')) \;\wedge\; (n \not\xrightarrow{\ell}_r n' \vee \exists m'' \cdot (\mathrm{Cons}(m'',n') \wedge m \xrightarrow{\ell}_p m''))\}$$

Figure 7: Definition of transitions for $M +_{pm} N$.

$$\begin{array}{rcl} ((m,n),\ell,(m',n')) \in \Delta_D & \Rightarrow & (((\forall m'',((m,n),\ell,(m'',n')) \in \Delta^r \Rightarrow ((m,n),\ell,(m'',n')) \in \Delta_D) \wedge \\ & & (\forall n'',((m,n),\ell,(m',n'')) \in \Delta_D \Rightarrow (n'=n'')) \wedge ((n,\ell,n') \in {\Delta^r}_N)) \vee \\ & & ((\forall n'',((m,n),\ell,(m',n'')) \in \Delta^r \Rightarrow ((m,n),\ell,(m',n'')) \in \Delta_D) \wedge \\ & & (\forall m'',((m,n),\ell,(m'',n')) \in \Delta_D \Rightarrow (m'=m'')) \wedge ((m,\ell,m') \in {\Delta^r}_M))) \end{array}$$

Figure 8: Formalization of condition (6) of Definition 14.

the original models. The boundary disagreement transitions in these models are

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

As in the merge operation in [13], 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 maximal size of the resulting pseudo-merge is $(|M|+1) \cdot (|N|+1)$ as we traverse pairs of states in $M \cup \{*\} \times N \cup \{*\}$.

Theorem 1. *$M +_{pm} N$ is a pseudo-merge.*

Proof of Theorem 1. Given two MTSs $M = \langle S_M, Act, s_M, \Delta^r_M, \Delta^p_M\rangle$ and $N = \langle S_N, Act, s_N, \Delta^r_N, \Delta^p_N\rangle$, let $P = \langle S_P, Act, p_0, \Delta^r_P, \Delta^p_P\rangle$ as defined in Definition 13. Let P_M be P's subMTS w.r.t. states $S \setminus I_M$. We show that $M \preceq P_M$. We define the relation $\mathcal{R} = \{(m,(m,n)) | (m,n) \in S \setminus I_M\}$ (n may well be $*$) and need to show that $\mathcal{R}$ is a refinement relation between P_M and M, that is, it satisfies both conditions of Definition 8. Let $((m,n),m) \in \mathcal{R}$. If $m \xrightarrow{\ell}_r m'$, then by one of the rules TT, TM, TF, or $\Gamma*$ with $\gamma = t$, there is a transition $(m,n) \xrightarrow{\ell}_r (m',n')$. Of all such transitions, at least one has not been removed by the removal rules (see Figure 7). On the other hand, if $(m,n) \xrightarrow{\ell}_p (m',n')$, this transition could not be created by rules FT or $*\Gamma$, since those lead to states in I_M. All other cases require that $m \xrightarrow{\ell}_p m'$. □

5.2 Correctness and Completeness

We now define the set of modal μ-calculus formula denoted by a $M +_{pm} N$, show that all the formulas in the set are distinguishing properties of M and N, and then that there are no distinguishing properties that provide shorter explanations to the inconsistency between M and N.

The set of modal μ-calculus formula denoted by a $M +_{pm} N$ is defined as the set of formula that can be constructed by any *distinguishing DAG* embedded into $M +_{pm} N$.

Definition 14 (Distinguishing DAG)**.** *Let $M +_{pm} N = \langle\langle S, L, \Delta^r, \Delta^p, s_0\rangle, I_M, I_N\rangle$. We call a DAG $D = (v_0, V, \Delta_D)$, where $V \subseteq S$ a* distinguishing DAG *iff*

(1) D has the same initial state, i.e., $v_0 = s_0$

(2) D contains only required transitions, i.e., $\Delta_D \subseteq \Delta^r$

(3) D contains no transitions from a disagreement state, i.e., $p \xrightarrow{\ell} p' \in \Delta_D \Longrightarrow p \notin (I_M \cup I_N)$

(4) Leaf transitions are disagreement, i.e., $p \in V \wedge (\forall p' \in S \cdot p \xrightarrow{\ell} p' \notin \Delta_D) \Longrightarrow p \in I_M \cup I_N$

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

(6) Each transition corresponds to taking all transitions on a symbol in one original MTS and only one, required, transition in the other, i.e., see Figure 8.

An example of a distinguishing DAG for $M +_{pm} N$ is the subgraph which consists of transitions $(0,0') \xrightarrow{propose}_r (4,4') \xrightarrow{accept}_r (5,5') \xrightarrow{applyLaw}_r (5,*')$ and $(0,0') \xrightarrow{propose}_r (4,1') \xrightarrow{amend}_r (0,*')$.

Definition 15 (Formula of a Distinguishing DAG)**.** *The* formula induced by a distinguishing DAG $D = (v_0, V, \Delta_D)$ *on the pseudo-merge* $\langle\langle S, L, \Delta^r, \Delta^p, (m_0, n_0)\rangle, I_M, I_N\rangle$ *is a* $\mathcal{L}_\mu$ *formula* $\mathcal{F}$*, defined inductively below:*

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

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

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

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

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

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

The formula for the distinguishing DAG in our example is $\phi = \langle propose\rangle(\langle accept\rangle\langle applyLaw\rangle\mathbf{t} \wedge \langle amend\rangle\mathbf{t})$. Note that projecting this distinguishing DAG onto the model $\mathcal{A}$ in Figure 1, we obtain the diagram of Figure 2(b) (see Section 2), where the dashed edges correspond to those covered by the DAG.

Definition 16 (Formulas of $+_{pm}$)**.** *The set of formulas of a pseudo-merge* $M +_{pm} N$ *are those induced by all distinguishing DAGs of* $M +_{pm} N$*.*

Theorem 2 (Correctness of $+_{pm}$)**.** *All formulas of* $M +_{pm} N$ *are distinguishing properties for* M *and* N*.*

The proof consists in proving the following lemma:

Lemma 1. *Let* D *be a distinguishing DAG on* $P = M +_{pm} N$ *and* $\varphi = \mathcal{F}(D)$*. Then* $M \vDash \varphi \wedge N \vDash \neg\varphi$*.*

Proof. 1. If D has only one transition, then it is between a normal (m,n) and a disagreement state (case (4) of Definition 14). This transition can only have been created by rules TF or FT.

If the rule TF 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 15, there is a transition $M \xrightarrow{\ell}_r M'$, and $M \vDash \varphi$. Since we know that there is no transition on ℓ from N, $N \vDash \neg\varphi$.

If rule FT was applied, we have a dual situation. The formula we obtain is $[\ell]\mathbf{f}$ (case (ii) of Definition 15), which is true in states of M that do not have a transition on ℓ and false in those in N that require one.

2. Suppose the lemma holds on all DAGs with up to a height k. We prove it for a DAG D with height $k+1$. Consider all transitions stemming from the root (M,N) of D. By (6) of Definition 14, we have two dual cases.

Suppose they are all transitions on ℓ from (M,N) to states $(M', N'_1), \ldots, (M', N'_k)$. By case (iii) of Definition 15, the formula φ is then $\langle\ell\rangle \bigwedge_{i=1}^{k} \varphi_i$, where φ_i is a shortand for $\mathcal{F}((m', n'_i))$. By induction

hypothesis, $\forall i \in \{1,\ldots,k\}, M' \vDash \varphi_i \wedge N'_i \vDash \neg\varphi_i$. In that case, we also know that there is a transition $M \xrightarrow{\ell}_r M'$. Therefore, $M \vDash \varphi$. With rules TT and TM, the operator builds a transition $(M,N) \xrightarrow{\ell}_r (M',N')$ for *every* possible transition on ℓ from N. Therefore, $\forall N \xrightarrow{N}_p {}', \exists i \in \{1,\ldots,k\}$ such that $N' = N'_i$. Since for all possible targets on ℓ from N the conjuction of the φ_is does not hold, $N \vDash \neg\varphi$.

The dual case is treated with case (iv) of Definition 15.

All formulas induced by the pseudo-merge are built from a distinguishing DAG, and are therefore *true* in M and *false* in N. □

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 any property that distinguishes M from N results in an explanation that is at least as long as some explanation derived from $M +_{pm} N$. By an *explanation* we mean a projection of proof trees of a distinguishing formula onto the models being compared.

Theorem 3 (Completeness of $+_{pm}$)**.** *Let M and N be inconsistent MTSs. For any distinguishing property φ and two proofs, Π^{φ}_{M} and $\Pi^{\neg\varphi}_{N}$ of the fact that $M \vDash \varphi$ and $N \vDash \neg\varphi$, there exists a formula φ' induced by $M +_{pm} N$, a proof $\Pi^{\varphi'}_{M}$ of $M \vDash \varphi'$ and a proof $\Pi^{\neg\varphi'}_{N}$ $N \vDash \neg\varphi'$ such that the projection of $\Pi^{\varphi'}_{M}$ on M (resp. $\Pi^{\neg\varphi'}_{N}$ on N) is subgraphs, with the same initial state, of the projection of Π^{φ}_{M} on M (resp. $\Pi^{\neg\varphi}_{N}$ on N).*

Outline of a proof of completeness (Theorem 3). The entire proof is available in Appendix A.1.

We start by combining the given proofs in one sole tree. We then collapse some nodes of this tree that correspond to loops in the pseudo-merge, and cut some superfluous subtrees. This results in another tree, encoding the (smaller) proof of another formula. We show that we can project this tree onto the pseudo-merge, and that the projection is a distinguishing DAG. Moreover, we show that the formula 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 ones. □

Finally, if M and N are consistent, then $M +_{pm} N$ yields a common refinement of M and N. In other words, $M +_{pm} N = \langle P, I_M, I_N \rangle$ s.t. $I_M = I_N = \emptyset$. Furthermore, P is the same model as the one returned by $+_l$ in [13].

Property 2 ($+_{pm}$ of consistent MTSs)**.** *If M and N are consistent, then $M +_{pm} N = M +_l N$.*

Proof. The only application of rules TM, MT, TT of $+_{pm}$ that have no direct correspondence in $+_l$ are those where succesor states, M' or N', are inconsistent, but these transitions will be removed because for each consistent (m,n), there is a consistent state (m'',n''). Conversely, as seen in Figures 4 and 6, all rules of $+_l$ are included in rules of $+_{pm}$. □

6 User-guided Feedback Generation

Having explained how sound graphical feedback on inconsistency can be provided from a property that distinguished two behavior models (see Section 4) and how a compact representation of all explanations to inconsistencies can be constructed (see Section 5), we now show how such a representation can be used by a modeler to explore the inconsistencies between two models.

A user who wishes to explore the reasons why two models are inconsistent can use the pseudo-merge operator to construct a model with which to explore the behaviour for which the two models being compared agree upon and to produce queries on why certain transitions to disagreement states of the pseudo-merge represent inconsistencies. The queries on disagreement transitions can be used to generate sound graphical feedback on why the transition partially represents an inconsistency between the models.

We now discuss how user-guided feedback on inconsistent models can be generated automatically. Specifically, we discuss an algorithm which given the pseudo-merge of two models M and N and a disagreement transition of the pseudo-merge produces a distinguishing DAG (recall Definition 14) over

the pseudo-merge which covers the disagreement transition. This disagreement DAG encodes (recall Definition 15) a property ϕ for M and N for which there exists proof trees showing that ϕ holds in M and $\neg\phi$ does not hold in M. These proof trees when projected on M and N cover the disagreement transition selected by the user. In short, 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.

Definition 17 ($S \sqcup T$)**.** *If S and T are sets of pairs of the form (a, B), with B being a set we define the following set*

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

Algorithm 1 BuildDistinguishingDAG

procedure BuildDistinguishingDAG(M_0, N_0)
  Let $P = M_0 +_{pm} N_0$
  Choose $(M_1, N_1) \xrightarrow{\ell}_r (M'_1, N'_1) \in P$ such that $(M'_1, N'_1) \in I_{M_0} \cup I_{N_0}$ (∗ Chosen by the user ∗)
  Let π be a strict[1] *required agreement* path to (M_1, N_1) traversing only pairs of inconsistent states
  $\pi = \pi :: ((M_1, N_1) \xrightarrow{\ell}_r (M'_1, N'_1))$ (∗ Extend π with our goal transition ∗)
  $X = \emptyset$ (∗ Set of visited states ∗)
  Let $I = \{(M, N), \{(M, N)\}\}$ (∗ Pairs of states to traverse first ∗)
  Let $J = \emptyset$ (∗ Pairs of states to traverse ∗)
  Let $D = (V, E) = (\{(M, N)\}, \emptyset)$ (∗ A placeholder for the DAG ∗)
  while $I \cup J \neq \emptyset$ **do**
    if $I = \emptyset$ **then**
      Let $(M, N), F = J$.pop()
    else
      Let $(M, N), F = I$.pop()
    $X = X \cup \{(M, N)\}$
    if $(M, N) \xrightarrow{a}_r (M', N')$ is a step in π **then**
      (∗ Try to apply the normal algorithm with letter a ∗)
      Let success = SubRoutine($M, N, F, \{M'\}, \{N'\}, \{a\}$)
      if ¬ success **then**
        Backtrack
    else
      Let success = SubRoutine($M, N, F, States(M), States(N), Act$) (∗ Apply the normal algorithm ∗)
      if ¬ success **then**
        Backtrack
  return D

Property 3 (Correctness of BuildDistinguishingDAG (Algorithm 1))**.** *If M and N are inconsistent MTSs over the same alphabet, then algorithm* BuildDistinguishingDAG *returns a distinguishing DAG of $M +_{pm} N$ containing the transition $(M_1, N_1) \xrightarrow{\ell}_r (M'_1, N'_1) \in P$, if there is one such DAG.*

Proof of property 3. We shall show that the graph that is built by BuildDistinguishingDAG is indeed a distinguishing DAG, as defined by Definition 14. 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. On each case of the SubRoutine, every transitions we add correspond to a required transition in at leat one of the models. Therefore, by one of the rules

Algorithm 2 SubRoutine

1: **procedure** SubRoutine($M, N, F, targets_M, targets_N, subAct$)
2: **if** $M \xrightarrow{a}_r M' \ \wedge \ N \not\xrightarrow{a} \ \wedge M' \in targets_M \ \wedge \ a \in subAct$ **then**
3: $V = V \cup \{(M', *)\}$
4: $E = E \cup (M, N) \xrightarrow{a}_r (M', *)$
5: **return** True
6: **else if** $N \xrightarrow{a}_r N' \ \wedge \ M \not\xrightarrow{a} \ \wedge N' \in targets_N \ \wedge \ a \in subAct$ **then**
7: $V = V \cup \{(*, N)\}$
8: $E = E \cup (M, N) \xrightarrow{a}_r (*, N')$
9: **return** True
10: **else if** $M \xrightarrow{a}_r M' \wedge (\forall N', N \xrightarrow{a}_p N' \Rightarrow \neg\text{Cons}(M', N') \wedge (M', N') \notin F) \wedge M' \in targets_M \wedge a \in subAct$ **then**
11: Let $T = \{N' | N \xrightarrow{a}_p N'\}$
12: $V = V \cup \{(M', N') | N' \in T\}$
13: $E = E \cup \{(M, N) \xrightarrow{a}_r (M', N') | N' \in T\}$
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.`remove`(N')
17: $J = (J \sqcup \{((M', N'), F \cup \{(M', N')\}) | N' \in T\}) \setminus X$
18: **return** True
19: **else if** $N \xrightarrow{a}_r N' \wedge (\forall M', M \xrightarrow{a}_p M' \Rightarrow \neg\text{Cons}(M', N') \wedge (M', N') \notin F) \wedge N' \in targets_N \wedge a \in subAct$ **then**
20: Let $T = \{M' | M \xrightarrow{a}_p M'\}$
21: $V = V \cup \{(M', N') | N' \in T\}$
22: $E = E \cup \{(M, N) \xrightarrow{a}_r (M', N') | N' \in T\}$
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.`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

TT, TM, MT, TF, FT, the transition we are adding to the DAG will be required. Because we never add states from $I_{M_0} \cup I_{N_0}$ in I or J, we will never take transitions that stem from one of these in the DAG. On the other hand, a transition will be a leaf only if it has been built by on of the two first cases of SubRoutine: otherwise, we will 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 that 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 ensure that from a state we add to the DAG all transitions on a letter for one of the MTS and only one from the other, and that this transition is required. Therefore, the graph returned by BuildDistinguishingDAG is a distinguishing DAG □

Applying the algorithm successively on disagreement transitions $(5,5') \xrightarrow{applyLaw}_r (5,*')$, $(4,1') \xrightarrow{amend}_r (0,*')$, $(5,5') \xrightarrow{applyAct}_r (*,5')$, and $(1,4'), \xrightarrow{amend}_r (*,0')$ of the pseudo-merge of Figure 3, we obtain distinguishing graphs corresponding to the following formulas.

$$\langle propose\rangle(\langle accept\rangle\langle applyLaw\rangle\mathbf{t} \wedge \langle amend\rangle\mathbf{t}) \tag{1}$$
$$\langle propose\rangle(\langle accept\rangle\langle applyLaw\rangle\mathbf{t} \wedge \langle amend\rangle\mathbf{t}) \tag{2}$$
$$\langle propose\rangle(\langle accept\rangle[applyAct]\mathbf{f} \wedge \langle amend\rangle\mathbf{t}) \tag{3}$$
$$[propose](\langle accept\rangle\langle applyLaw\rangle\mathbf{t} \vee [amend]\mathbf{f}) \tag{4}$$

Note that the formulas derived from the DAGs that cover the first two transitions ($(5,5') \xrightarrow{applyAct}_r (*,5')$ and $(4,1'), \xrightarrow{amend}_r (0,*')$) are identical. This is because, both transitions form part of exactly the same argument as to why the models are inconsistent.

As mentioned above, there are cases when it is not possible to construct a distinguishing DAG that covers the user-selected disagreement transition. An example of such a case is shown on Figure 9 where if transition $(3,2') \xrightarrow{d}_r (*,3')$ is selected, there is no distinguishing DAG over the pseudo-merge that covers it. This is because if we want to reach this transition, we have first to take transition $(0,0') \xrightarrow{a}_r (2,1')$, but condition (6) of Definition 14 states that we have to take all possible transitions on a from state 0. That forces us to have transition $(0,0') \xrightarrow{a}_r (1,1)$ in the DAG. Since condition (4) of the definition of a distinguishing DAG forces us to continue from $(1,1)$ which is not a disagreement state, we must include transition $(1,1) \xrightarrow{b}_r (0,0')$ in the DAG, which forms a loop, violating condition (1) of the definition, that requires that we have an acyclic graph.

The reason for the non-existence of a distinguishing DAG that covers $(3,2') \xrightarrow{d}_r (*,3')$ is that distinguishing properties that cover the 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 has proof trees that are strictly larger than formula $\langle b\rangle\mathbf{t}$ which corresponds to the formula generated by the algorithm when transition $(0,0) \xrightarrow{b}_r (4,*')$ is selected. Informally, the fact that we can reduce a distinguishing formula covering $(3,2') \xrightarrow{d}_r (*,3')$ into one does cover the transition shows that this disagreement is not relevant and that only the one that is relevant is represented by $(0,0) \xrightarrow{b}_r (4,*')$. The reason why this irrelevant disagreement transition appears in the pseudo-merge is that the rule for removing unnecessary transitions for the non-deterministic case is slightly week. We aim to refine the construction of pseudo-merge to eliminate these borderline cases.

Summarising, in this section we have shown how to build a distinguishing DAG that covers a user-selected disagreement transition in a pseudo-merge. As shown in previous sections, resulting DAG can be used to provided sound feedback on the inconsistencies between the models being compared. We have shown that the psuedo-merge may include some disagreement transitions, related to non-deterministic choices, that do no yield minimal distinguishing properties. The practical implications of this are that the user, when selecting one of these transitions, may be required to pick a different transition in order to produce feedback on inconsistency.

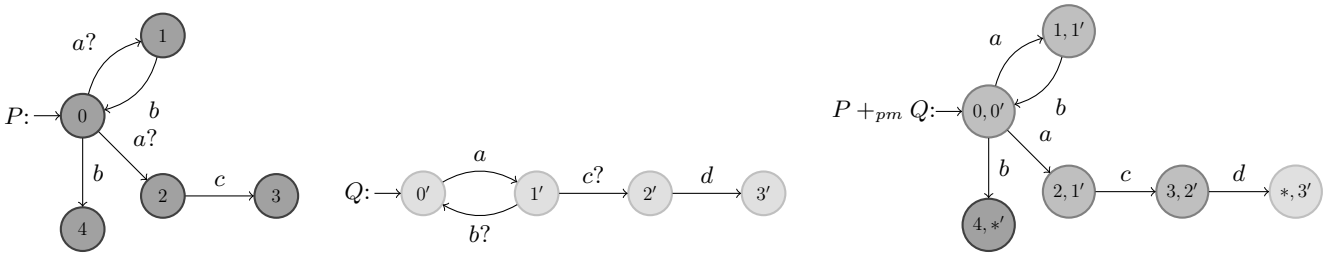


Figure 9: A case when we cannot build a distinguishing DAG containing a given transition

7 Validation

In this section we present several models that have been used to test and validate our approach. The first one uses small models of a printer, the second is a bigger one of a safety injection system, adapted from [9].

7.1 The special case of LTSs

Although presented in the more general case of MTSs, our thechnique is also appliable for LTSs. If we see MTSs as the set of LTSs that implement it, the only implementation of an LTS being itself modulo bisimulation, and consistency as the fact of having a common refinement, consistency between LTSs is exactly the same as bisimilarity. In this case we would therefore produce a graphical explanation of why the models are not bisimilar.

7.2 The printer case study

To build this example, we asked two grad-students/coworkers/fellow students to act as different stakeholders that are building the same model. We gave them an alphabet and its intended semantics in english, and asked them to build independently a state-machine modfelling a printer with a pre-output tray where sheets are stapled. The rest of the requirements where willingly imprecise to force them to make choices that could be inconsistent.

The specification of the printer as it was given originally is the following.

- 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

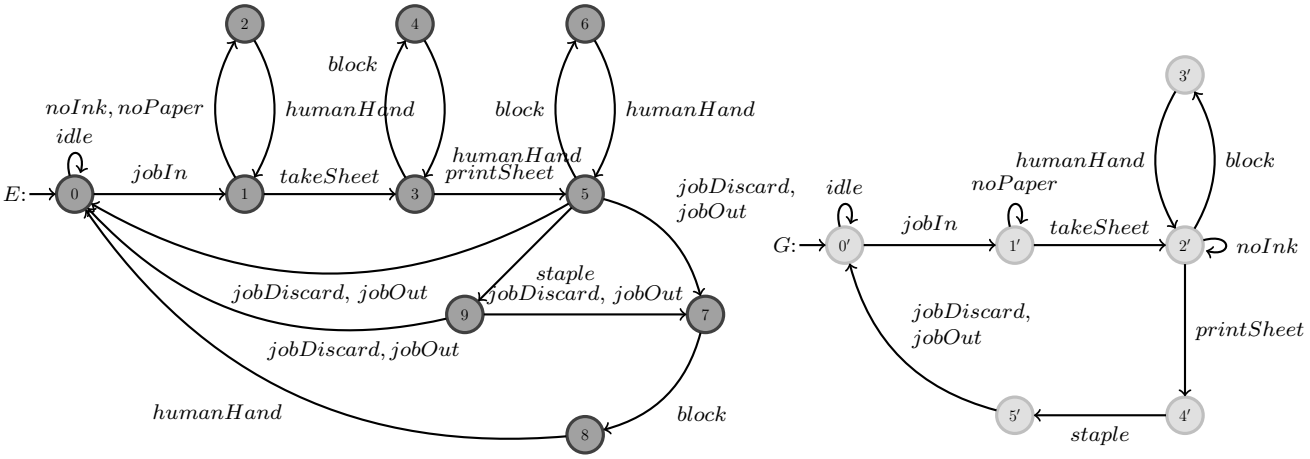
  takeSheet The rolls take a sheet from the tray

  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



(a) Esteban's model

(b) Guido's model

Figure 10: Models of a printer drawn by Esteban and Guido.

noInk No more ink in the toner

humanHand A repairman fixes the printer

- Requirement: A sheet never goes backwards

  1. on the paper tray
  2. taken
  3. printed
  4. goes on the pre-output tray
  5. goes on the output tray or garbage

The models that where produced are shown on Figure 10. We then applied our $+_{pm}$ operator to these models. The result is depicted on Figure 11. We can see that there is a main part on which the models agree:the big loop $(0,0') \xrightarrow{jobIn}_r (1,1') \xrightarrow{takeSheet}_r (3,2') \xrightarrow{printSheet}_r (5,4') \xrightarrow{staple}_r (9,5') \xrightarrow{jobOut}_r (0,0')$ (the last action can also be a *jobDiscard*). Incidentally, this part corresponds to an execution without error of the printer. On the pseudo-merge we can identify 13 transitions from agreement to disagreement states. From each of these transitions we can try to build a distinguishing DAG, and therefore a formula that distinguishes between models E and G. We obtain 11 distinct distinguishing formulas (because the formula does not take into account the target of the leaf disagreement transition). Note that although model E is non deterministic, none of the formula explicitely use this non determinism. For example, consider formula $\langle jobIn\rangle\langle takeSheet\rangle\langle printSheet\rangle\langle jobOut\rangle\mathbf{t}$ that corresponds to one of the transition on *jobOut* from state 5 that does not have a counterpart in model G since it requires to staple the sheets first. On the other hand, the way errors are handled in both models is different, and that yields totally different inconsistencies, for example the formula $\langle jobIn\rangle\langle noPaper\rangle[takeSheet]\mathbf{f}$ that corresponds to the fact that in model E only a human intervention can solve the lack of paper, whereas in model G the problem "solves itself".

7.3 The safety injection system

The safety injection system is part of a controller for a nuclear power plant. It is suppose 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. It was presented in [9] where the global goal is refined into smaller ones used to generate a model. By changing one of these goals, we generate another model, that is not bisimilar to the original one. The size of these models (25 and 48 states) does not allow us to represent them here. The change we made is to require a delay of one time unit (*tick*) before starting or

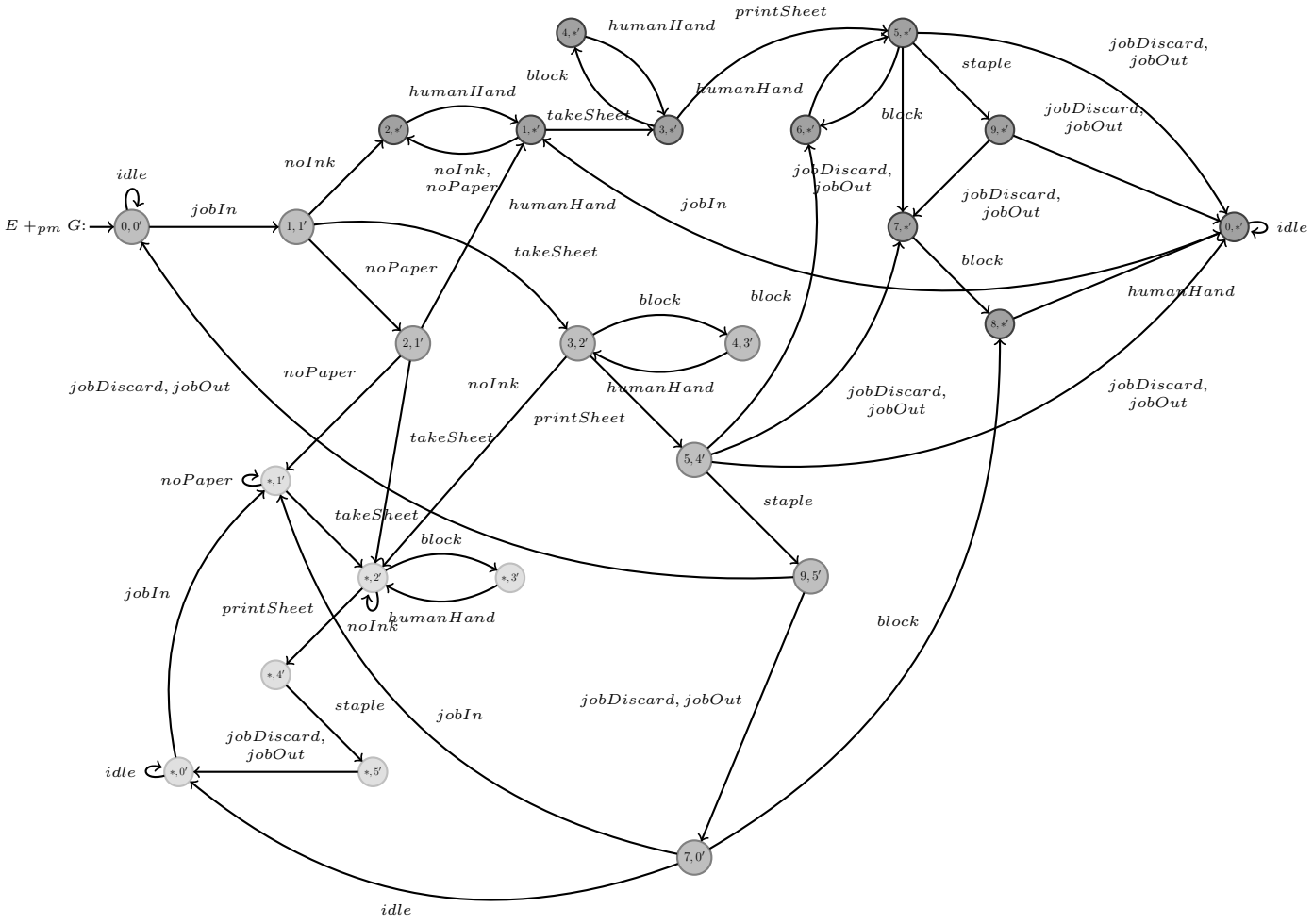


Figure 11: The pseudo-merge of the printer models of Figure 10.

stopping the injection system when this is required, that is to say when we are in a normal mode with a low pressure of coolant in the system. We introduced non-determinism in the models before comparing them by removing the information we had on the values of the pressure when it changes. For example, in both models we replaced all transitions on an action $raisePressure[x]$ or $lowerPressure[x]$ by an action $pressure$, thus hiding the value of x. By building a pseudo-merge (not represented for it has 220 states) and generating distinguishing formulas, we obtain for example

$$\begin{aligned}\psi \;=\; & \langle enableSafetyInjection\rangle\langle tick\rangle\langle toc\rangle\langle pressure\rangle \\ & (\langle sendSafetyInjectionSignal\rangle\langle tick\rangle\langle toc\rangle\langle stopSafetyInjectionSignal\rangle\langle tick\rangle\mathbf{t} \\ & \wedge\ [tick]\mathbf{f})\end{aligned}$$

We can see that for the *pressure* transition in the first model, there is no good corresponding transition in the second model: one leads to an inconsistency we introduced by requiring a delay of one *tick* before strating the system; the other one leads to parts of the models where the pressure is assumed to be different (but was hidden), finally leading to an inconsistency.

8 Conclusion and future work

We have defined a formalism to represent inconsistencies between two MTSs in one augmented model we call a pseudo-merge. We have presented an way to build such a model through our $+_{pm}$ operator. The obtained model contains several explanations of why the models are inconsistent. Moreover, all explanation of inconsistency can be reduced to the ones contained in our pseudo-merge. We can therefore explore inconsistencies between the models, keeping the explanations as small as possible. However, the pseudo-merge may contain misleading information, as some inconsistencies can trigger the display of consistent parts of the models as inconsistent. We hope to refine our construction of the pseudo-merge in order to get rid of such cases.

References

[1] M. Chechik, N. D'Ippolito, D. Fishbein, and S. Uchitel. "MTSA: The Modal Transition System Analyzer". In *Proceedings of International Conference on Automated Software Engineering (ASE'08)*, 9 2008.

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

[3] E. 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, Copenhagen, Denmark, July 2002. IEEE Computer Society.

[4] D. Fischbein and S. Uchitel. "On Correct and Complete Merging of Partial Behaviour Models". In *Proceedings of ACM SIGSOFT Sixteenth International Symposium on the Foundations of Software Engineering (FSE'08)*, 11 2008.

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

[6] 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.

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

[8] K. 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.

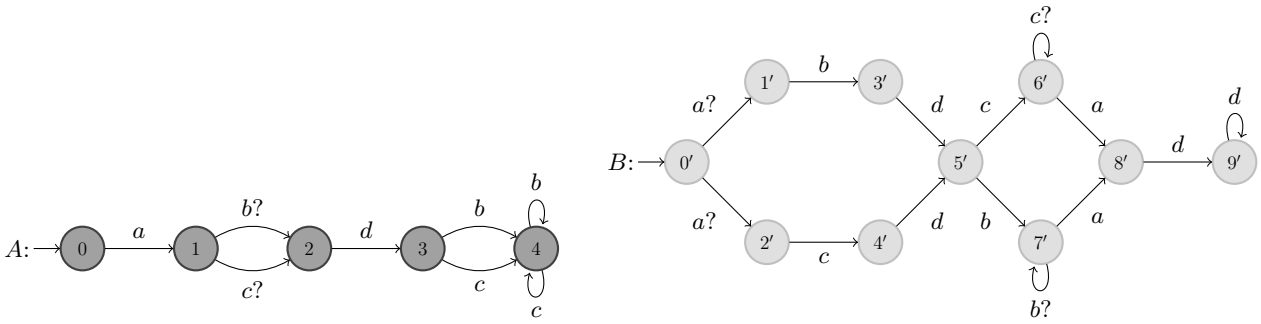
[9] E. Letier and S. Uchitel. "Deriving event-based transition systems from goal-oriented requirements models". In *Proceedings of International Conference on Automated Software Engineering (ASE'08)*, 9 2008.

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

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

[12] C. Stirling. "Modal and Temporal Logics for Processes". In *Proceedings of the VIII Banff Higher Order Workshop Conference on Logics for Concurrency : Structure Versus Automata*, pages 149–237. Springer-Verlag New York, Inc., 1996.

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



(a) (b)

Figure 12: Models used to illustrate completeness of the pseudo-merge.

A Proof of selected properties

A.1 Proof of completeness of the pseudo-merge

In this section we shall prove Theorem 3. Since it is a rather long but constructive proof, we shall illustrate each step with an example.

The proof procedes as follows. We start by combining the given proofs in one sole tree. We then collapse some nodes of this tree that correspond to loops in the pseudo-merge, and cut some superfluous subtrees. This will give us another tree that encodes the proof of a new formula, smaller. We will show that we can project this tree onto the pseudo-merge, and that the projection is a distinguishing DAG. Moreover, we will show that the formula induced by the distinguishing DAG is the same that is proved by the tree. Finally, we will split the tree into two proof-trees, one for each model. The way the new tree is built will ensure that the new proofs are projected as subgraphs of the original ones.

Example The example we are going to use are the models A and B of Figure 12. We will suppose that the distinguishing property we originally have is $\varphi = \langle a\rangle([b]\langle d\rangle\langle c\rangle\langle c\rangle[a][d]\mathbf{f} \wedge [c]\langle d\rangle\langle b\rangle[a]\mathbf{f})$. Proof-trees for this property on the models are shown on Figure 13. We don't show the projection of these proof-trees on the original models since in both cases it corresponds nearly to the whole model. In A, the only transition not included in this projection are the self-loop on b on state 4. In B, self loop on $b?$ on state $7'$ is the only one left out.

A.1.1 Combining the proof-trees

First, let us define a combination $\Pi^{\varphi}_{M,N}$ of the proof trees Π^{φ}_{M} and $\Pi^{\neg\varphi}_{N}$. Intuitively, $\Pi^{\varphi}_{M,N}$ will encode 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 18 (The *Comb* operator)**.** *Given two proof-trees for a formula and its negation on two models, 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* $((*, n), \varphi')$

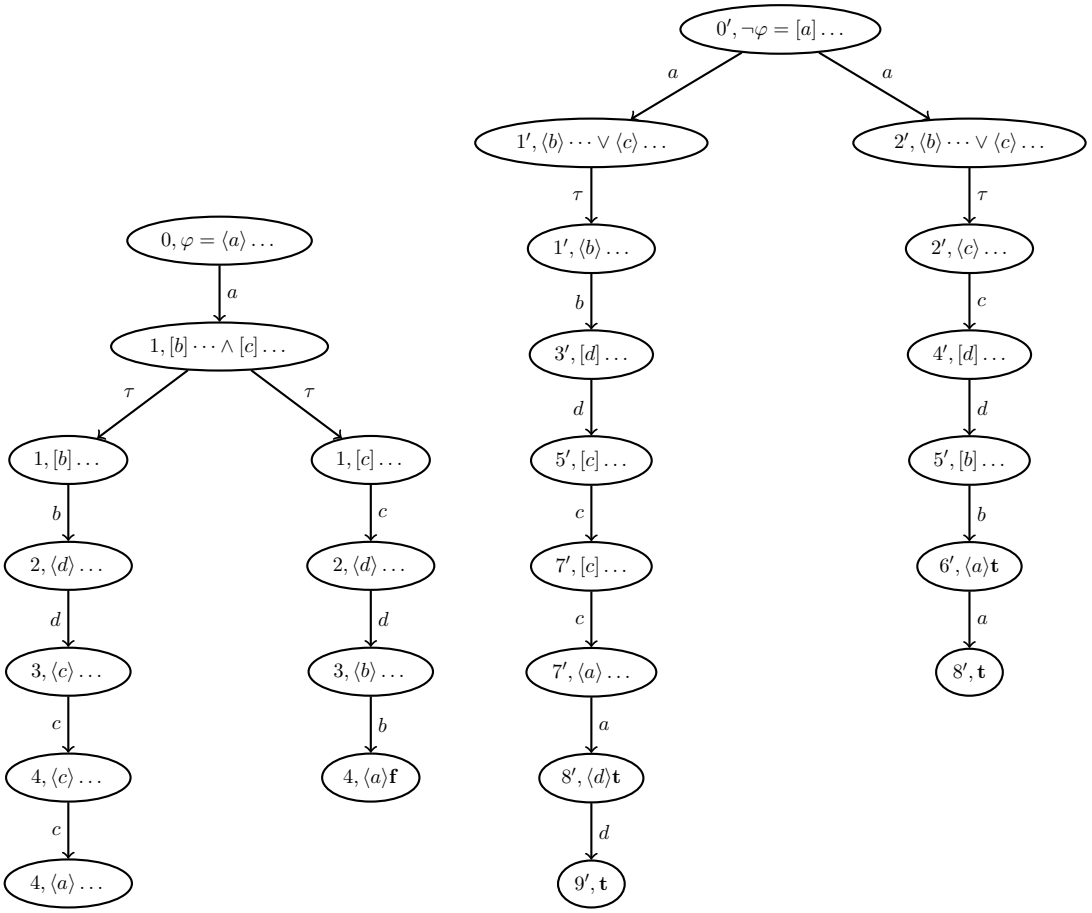
(iii) $Comb((M, \langle\ell\rangle\varphi) \xrightarrow{\ell} (M', \varphi), (N, [\ell]\neg\varphi)) = ((M, N), \langle\ell\rangle\varphi) \xrightarrow{\ell} Comb((M, *), \varphi)$

(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) $Comb((M, \langle\ell\rangle\varphi) \xrightarrow{\ell} (M', \varphi), (N, [\ell]\neg\varphi) \xrightarrow{\ell} \{(N'_i, \neg\varphi)\}_i) =$
$((M, N), \bigwedge_{i=1}^{k}\varphi) \xrightarrow{\ell} \{Comb((M', \varphi), (N'_i, \neg\varphi))\}_i$

(vi) $Comb((M, [\ell]\varphi) \xrightarrow{\ell} \{(M'_i, \varphi)\}_i, (N, \langle\ell\rangle\neg\varphi) \xrightarrow{\ell} (N', \neg\varphi)) =$
$((M, N), \bigvee_{i=1}^{k}\varphi) \xrightarrow{\ell} \{Comb((M'_i, \varphi), (N', \neg\varphi))\}_i$

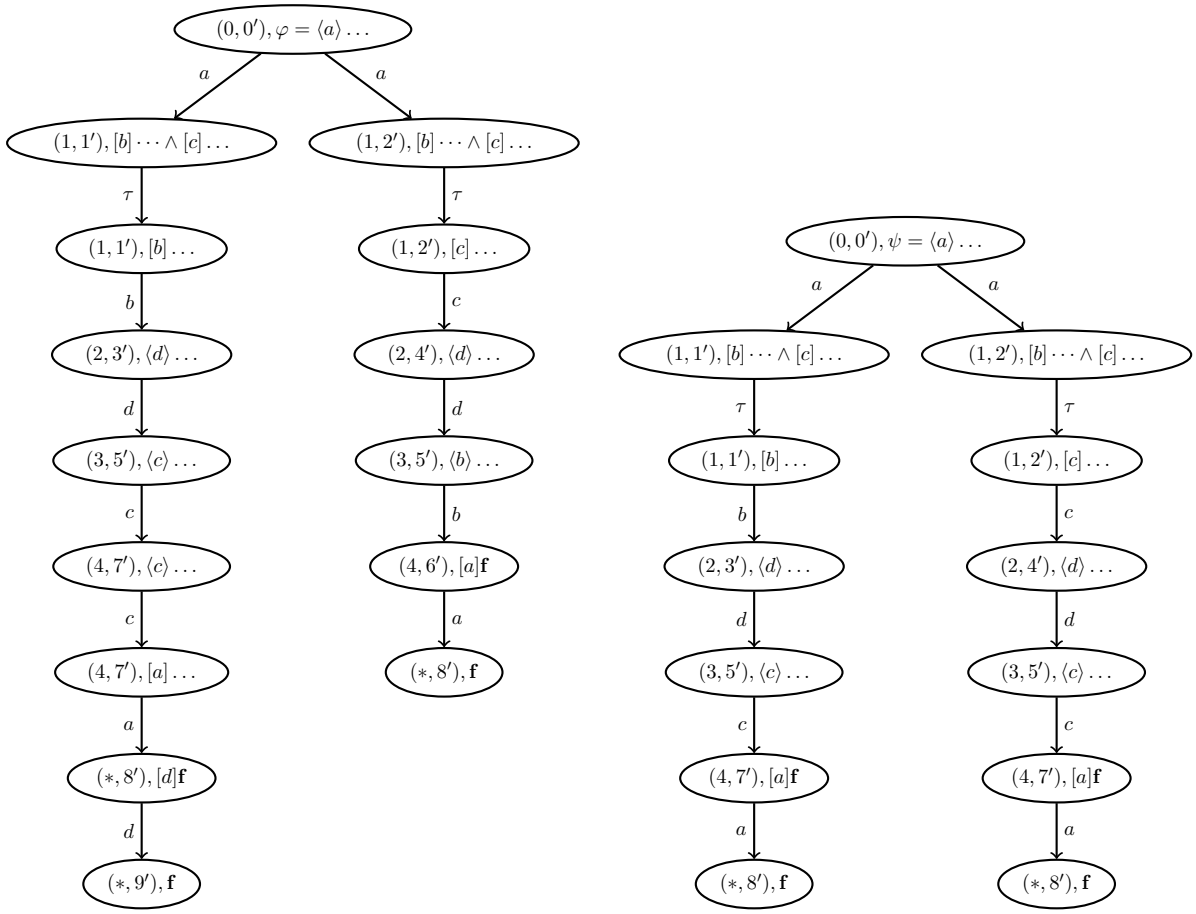
(vii) $Comb((M, \bigwedge_{i=1}^{k}\varphi_i) \xrightarrow{\tau} \{(M, \varphi_i)\}_i, (N, \bigvee_{i=1}^{k}\neg\varphi_i) \xrightarrow{\tau} (N, \neg\varphi_{i_0})) =$
$((M, N), \bigwedge_{i=1}^{k}\varphi_i) \xrightarrow{\tau} Comb((M, \varphi_{i_0}), (N, \neg\varphi_{i_0}))$



(a) A proof-tree of $A \vDash \varphi$

(b) A proof-tree of $B \vDash \neg\varphi$

Figure 13: The original proof-trees showing that $\varphi = \langle a \rangle([b]\langle d \rangle\langle c \rangle\langle c \rangle[a][d]\mathbf{f} \wedge [c]\langle d \rangle\langle b \rangle[a]\mathbf{f})$ is *true* in A and *false* in B.



(a) Combination of the proof-trees of Figure 13. τ transitions between identical nodes have been ignored to reduce the size of the tree.

(b) The new tree obtained by reducing the one on Figure 14(a)

Figure 14: Combination of the proof-trees of Figure 13 and a reduction of it.

(viii) $Comb((M, \bigvee_{i=1}^{k} \varphi_i) \xrightarrow{\tau} (M, \varphi_{i_0}), (N, \bigwedge_{i=1}^{k} \neg\varphi_i) \xrightarrow{\tau} \{(N, \neg\varphi_i)\}_i) =$
$((M,N), \bigvee_{i=1}^{k} \varphi_i) \xrightarrow{\tau} Comb((M, \varphi_{i_0}), (N, \neg\varphi_{i_0}))$

Rules (v) and (vi) of the previous definition duplicate the subformula φ, but the obtained formula is logically equivalent to the original one. We assume that we treat each copy separately. In particular, we assume that we keep track of which duplicate corresponds to which child. Note that we actually lose some information in cases (vii) and (viii). But the information is either not really important (in the sense that it represents a proof for a part of the formula that will go away) or not really lost but present in a sibling, and can be retreived 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 node there is a node $(m, \varphi') \in \Pi_M^{\varphi}$ (resp. $(n, \neg\varphi') \in \Pi_N^{\neg\varphi}$).

Example The combination of proof-trees of Figure 13 yield the combined proof-tree of Figure 14(a). If we project it on the pseudo-merge of models A and B displayed on Figure 15, we get almost all the graph (there again, only self loop on b on state $(4,7')$ is not included in the projection). We can notice that we don't have a DAG because of the loop on state $(4,6')$, that from the pair of states $(3,5')$ there are outgoing transitions on letters b and c, and that there is a transition stemming from a disagreement state (on a, from state $(*,8)$).

A.1.2 Reducing the combination

Now we will remove from this tree parts of the formula, and hence of the proof, that are not crucial to show the essence of the explanation of the inconsistency. Some of these parts are loops. If two states are inconsistent and there are two ways of showing it, in terms of two formulas φ and φ' and their proofs, with φ' being a subformula of φ, the witness φ' is a smaller therefore better one. Similarly, the witness

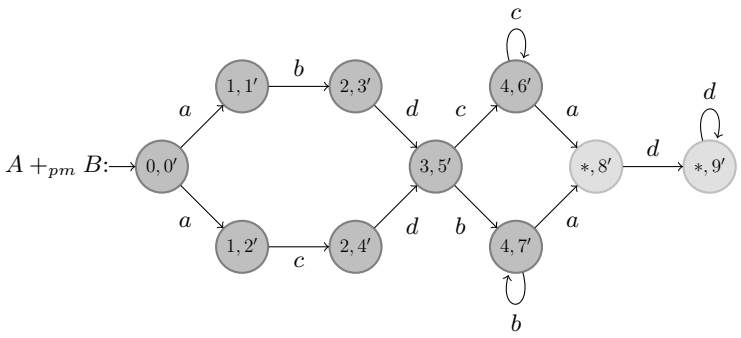


Figure 15: Pseudo-merge of the models of Figure 12

does not need to be too long, in the sense that once one of the proof-tree being combined has reached a leaf and the other hasn't, we have a witness of inconsistency. More formally, we apply the following procedure to $\Pi^{\varphi}_{M,N}$:

Step 1 Replace each highest node $((m, *), \varphi') \longrightarrow \dots$ by $((m, *), \mathbf{t})$, and each highest node $((*, n), \varphi') \longrightarrow \dots$ by $((*, n), \mathbf{f})$.

Step 2 For each node pair (m, n), find the set $F_{m,n}$ of all nodes of the form $((m, n), \varphi'')$. Select one of the lowest, that is to say a node $((m, n), \varphi'') \in F_{m,n}$ of which no descendent is in $F_{m,n}$ Then replace 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 $((m', n'), \varphi'')$, and if $m \xrightarrow{\ell}_r m' \in M$, then change $[\ell]\varphi'$ into $\langle\ell\rangle\varphi'$

Step 4 Propagate the changes made to the formulas bottom-up. Since we have kept track of which subformula corresponds to each child, if the child has changed, we change the corresponding subformula in the parent. Having made copies of the formula, a change in a subtree will not affect a sibling.

We obtain a new tree $\Pi^{\varphi'}_{(M,N)}$. Note that all τ transitions have been collapsed in step 2. Therefore all formulas will either be a constant ($\mathbf{t}$ or $\mathbf{f}$) or start by a $\langle\ell\rangle$ or a $[\ell]$. In the latter case, the outgoing transitions will bear the letter ℓ. Step 3's only purpose is to make an arbitrary choice between $\langle\ell\rangle$ and $[\ell]$ when it does not matter, that is to say when there is one and only one transition on ℓ from both states m and n, and that this transition is required. We shall later show that this tree also encodes the proofs for φ' on M and N, that is to say that we will be able to extract two proof-trees $\Pi^{\varphi'}_{M}$ and $\Pi^{\neg\varphi'}_{N}$ from it.

Example When we apply the procedure described above to our example, we start by cutting the branch $(4, 6'), [a][a]\mathbf{f} \xrightarrow{a} (*, 8'), [d]\mathbf{f} \xrightarrow{d} (*, 9')$ into $(4, 6'), [a]\mathbf{f} \xrightarrow{a} (*, 8'), \mathbf{f}$. Then we remark that the pair of states $(3, 5')$ appears twice (we do not count the trivial collapsing of τ transitions). Since the nodes labelled with this pair are not ancestor of one another, we can choose any of them to replace the other. Suppose that we choose to keep the subtree whose formula is $\langle c\rangle\langle c\rangle[a][a]\mathbf{f}$ (note that we have not yet updated the formula accordingly to the changes made to one of its branches). In that case we still have two occurences of the pair $(4, 6')$. In that case, the one labeled with $\langle c\rangle[a][a]\mathbf{f}$ is an ancestor of the one labeled by $[a][a]\mathbf{f}$. We will therefore keep the latter, and it will replace the former. Finally, we update the labels of the formulas, starting from the leaves, yielding the new tree of Figure 14(b). The formula at the root of this tree is $\psi = \langle a\rangle([b]\langle d\rangle\langle c\rangle[a]\mathbf{f} \wedge [c]\langle d\rangle\langle c\rangle[a]\mathbf{f}$. Note that the graph we obtain by projecting this tree on the pseudo-merge is a distinguishing DAG.

A.1.3 Having a distinguishing DAG

But for now, we will show that this formula is induced by the pseudo-merge. Let us consider the projection of $\Pi^{\varphi'}_{(M,N)}$ over $M +_{pm} N$. This projection is possible because each non-τ transition in the original tree $\Pi^{\varphi}_{(M,N)}$ corresponds to at least a required transition in an original model. By one of the rule of Figure 6 except MM, $\Gamma*$, or $*\Gamma$ when $\gamma = m$, there will be a required transition built in $M +_{pm} N$. Moreover,

none of these transitions will be removed, because this only happens when there is no inconsistency on the letter, and this is not the case since we have a distinguishing property starting with this letter. Since the transformation of $\Pi^{\varphi}_{(M,N)}$ into $\Pi^{\varphi'}_{(M,N)}$ collapses only nodes that share the same labels for states, and never adds transitions, we will still be able of projecting it on the pseudo-merge. As said before, this projection will only have required transitions, therefore complies with condition 2 of Definition 14. Moreover, step 2 of the transformation has removed all loops. Therefore we are sure that the projection will be a directed acyclic graph, as is required by condition 1. Step 1 has removed all transitions from nodes corresponding to $(m, *)$ or $(*, n)$. Hence the projection verifies condition 3. In addition, all leaves are either made so by step 1 or just copies from ones of an original proof by cases (iii) followed by an application of case (i) to the leaf or, dually, (iv) followed by an application of case (i) to the leaf. In all cases, the leaves are disagreement states, and condition 4 of Definition 14 holds. By step 2, all nodes corresponding to a pair of states will bear the same formula. As said before, the outgoing transitions from these will be the first letter of the formula. Since the formula is the same, the letter will be the same, and condition 5 will hold. Finally, in the case 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 definition of these, we had a required transition in M and consider only this one, whereas we have considered all possible transitions in N. By a dual reasonning in the case when the formula starts with $[\ell]$, condition 6 of Definition 14 holds. Therefore, we can project $\Pi^{\varphi'}_{(M,N)}$ on the pseudo-merge and obtain a distinguishing DAG.

A.1.4 Extracting a formula

Let us show that the formula we obtain from this DAG with the $\mathcal{F}$ operator is exactly φ'. By step 1 of the transformation, the cases (i) and (ii) of the tree with only one transition produces the same formula. Suppose that $\mathcal{F}$ does produce the formula that labels the nodes when applied to the projection of all subtrees. 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 conjuction of formulas, one for each child. Even if they 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 bears the formula that could have been produced by the $\mathcal{F}$ operator over the projection, so does the parent. Case (iv) being dual, nodes always bear the formula that would have been produced by the application of the $\mathcal{F}$ operator over the projection of $\Pi^{\varphi'}_{(M,N)}$ over $M +_{pm} N$. It is therefore the case for φ' itself. Hence φ' is a formula induced by the pseudo-merge.

A.1.5 Building new proof-trees

Let us now produce separate proof-trees $\Pi^{\varphi'}_{M}$ and $\Pi^{\neg\varphi'}_{N}$ from the tree $\Pi^{\varphi'}_{(M,N)}$. To that end we use the *Split* operator on our tree $\Pi^{\varphi'}_{(M,N)}$.

Definition 19 (The *Split* operator)**.** *For a tree of which nodes are labelled with a pair of states and a $\mathcal{L}_\mu$ formula, we inductively define the Split operator as follows ($\bot$ is the empty tree; transitions leading to it are ignored):*

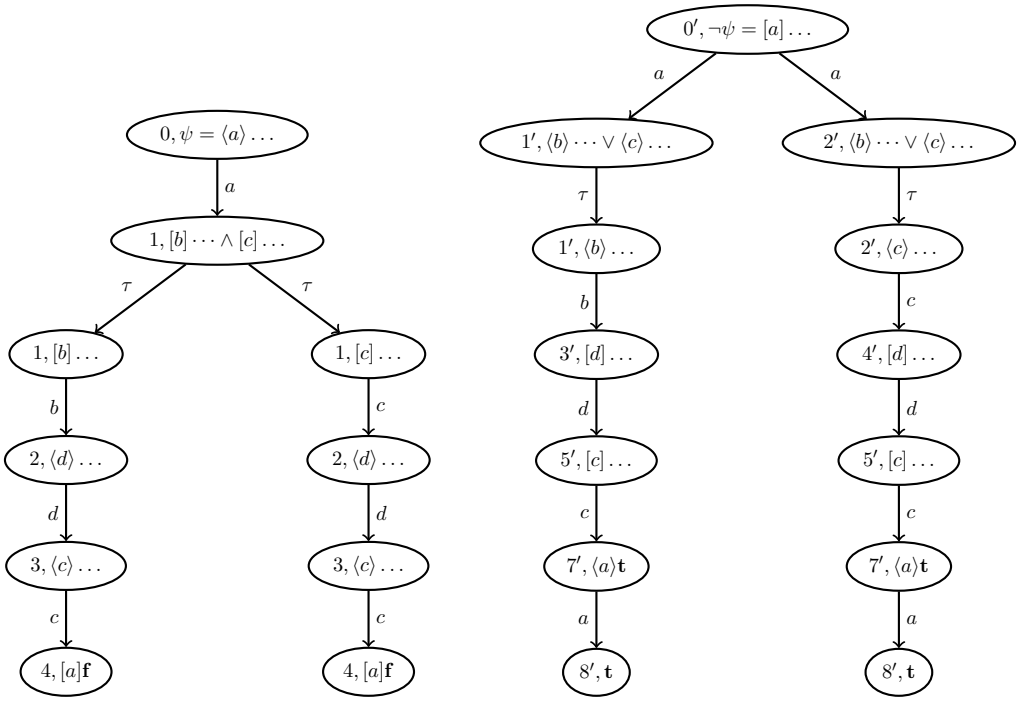
(i) $Split((M, *), \mathbf{t}) = ((M, \mathbf{t}), \bot)$

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

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

(iv) $Split(((M, N), [\ell] \bigvee_{i=1}^{k} \varphi_i) \xrightarrow{\ell} \{((M'_i, N'), \varphi_i)\}_i) =$
$((M, [\ell] \bigvee_{i=1}^{k} \varphi_i) \xrightarrow{\ell} \{(M'_i, \bigvee_{i=1}^{k} \varphi_i) \xrightarrow{\tau} \Pi^{\varphi_i}_{M'_i}\}_i, (N, \langle\ell\rangle \bigwedge_{i=1}^{k} \neg\varphi_i) \xrightarrow{\ell} (N', \bigwedge_{i=1}^{k} \neg\varphi_i) \xrightarrow{\tau} \{\Pi^{\varphi_i}_{N'}\}_i)$
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.



(a) A proof-tree for $A \vDash \psi$ (b) A proof-tree for $B \vDash \neg\psi$

Figure 16: The proof-trees that show that ψ is *true* in A and *false* in B, obtained from the reduced tree of Figure 14(b)

Note that the above definition is total in our case, because of the particular form of our tree. That is to say we will always be able to use one of the cases and build two trees. To prove that the trees produced by *Split* are indeed proof-trees, we just have to prove that on a $[\ell]$ (resp. $\langle\ell\rangle$) we do consider all possible transitions on ℓ in M (resp. N). This is true because we originally had proof-trees and never remove transitions only: nodes are replaced by others, that are also correct because they come from the tree. Therefore when we remove transitions, we remove them all, and change the formula accordingly.

Example When we split the tree of Figure 14(b), we obtain the proof-trees of Figure 16. We can remark that the projection of each of these new proof-trees is indeed a subgraph of the projection of the original corresponding proof-tree.

What remains to be shown is that the proofs we produced are subgraphs of the ones we initially had. Since we never add transitions and only replace nodes for ones with the same labels, all transition in 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).