A Characterization of Merging Partial Behavioural Models

by

Greg Brunet

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

A Characterization of Merging Partial Behavioural Models

Greg Brunet

Master of Science

Graduate Department of Computer Science

University of Toronto

2006

Constructing comprehensive operational models of intended system behaviour is a complex and costly task. Consequently, practitioners have adopted techniques that support partial behaviour description and focus on elaborating these descriptions iteratively. Scenario-based specifications, for example, are incrementally elaborated to cover system behaviour that is of interest. However, how should partial behavioural models described by different stakeholders with different viewpoints be composed? How should partial models of component instances of the same type be put together?

In this thesis, we use model merging based on observational refinement as a general solution to these questions, where merging consistent models is a process that results in a minimal common refinement. We prove several mathematical characterizations of merging and consistency, study algebraic properties of the merge operator, and give new and improved algorithms related to constructing merge. Finally, we present a case study that illustrates the utility of our results.

Acknowledgements

First, and foremost, I thank my advisor Marsha Chechik for her patience, enthusiasm, and ability to always see through a mess of questions and point me in the right direction. I also thank her for teaching me the ins and outs of the research process.

Second, I thank Sebastian Uchitel of Imperial College for spending a great amount of time with me on this work. He has taught me many things, particularly how to provide insight into the practicality of theoretical results, and how to keep the high-level picture in mind while studying the technical details.

Third, I thank Steve Easterbrook for agreeing to be my second reader, in light of his very busy schedule, and for his helpful comments.

Fourth, I thank for the members of the Formal Methods Group at U of T, especially Shiva Nejati, for helpful examples, always being available, and for spending significant amounts of time providing insights into my difficulties.

Last, but not least, I thank my Family and my girlfriend Alanna for their constant support and encouragement throughout this entire process. Without them I would not be here, and for that I am greatly indebted.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

State-based behaviour modelling and analysis has been shown to be successful in uncovering subtle design errors [8]. However, the adoption of such technologies by practitioners has been slow. Partly, this is due to the difficulty of constructing behavioural models – this task requires considerable expertise in modelling notations that developers often lack. In addition, and perhaps more importantly, the benefits of the analysis appear *after* comprehensive behavioural models have been built: classical state-based modelling approaches are generally not suited for providing early feedback, when system descriptions are still partial.

In contrast, interaction-based specifications, such as message sequence charts [30], are becoming increasingly popular. These scenario-based notations are partial behavioural descriptions that promote *incremental* elaboration of system behaviour.

There has been interest in developing an understanding and exploiting the relation between interaction-based and state-based modelling techniques [46]. In particular, several approaches to the synthesis of state-based models from scenario-based specifications (e.g. [51, 35]) have been developed. These approaches aim to combine the benefits of the incremental elaboration in interaction-based specifications with the behavioural analysis in state-based models.

A current limitation of synthesis approaches is that the models being synthesized, e.g., labelled transition systems (LTSs) [31], are assumed to be complete descriptions of the system behaviour up to some level of abstraction, i.e., the state machine is assumed to completely describe the system behaviour with respect to a fixed alphabet of actions. This completeness assumption is limiting, particularly if state-based modelling is to be adopted in iterative development processes [3], processes that adopt use-case and scenario-based specifications (e.g., [52]), or that are viewpoint-oriented [26].

In such development contexts, a more appropriate type of state-based model to synthesize is one in which currently unknown aspects of behaviour can be explicitly modelled. These models can distinguish between positive, negative, and unknown behaviours: positive behaviour refers to the behaviour that the system is expected to exhibit, negative behaviour refers to the behaviour that the system is expected to never exhibit, and unknown behaviour could become positive or negative, but the choice has not yet been made. State-based models that distinguish between these kinds of behaviour are referred to as *partial behavioural models*. A number of such models exist and promising results on their use to support incremental modelling and viewpoint analysis has been reported (e.g., Partial Labelled Transition Systems (PLTSs) [50], multi-valued state machines [12], Modal Transition Systems (MTSs) [37], Mixed Transition Systems [11] and multi-valued Kripke structures [5]).

1.1 Motivation for Merge

Although synthesis of partial behavioural models can provide substantial benefits [50], such models lack a specific concept that is particularly helpful in the context of behavioural model elaboration, namely, *model merging*. Scenarios are typically provided by different stakeholders with different viewpoints [26], describing different, yet overlapping aspects [6] of the same system. How should these partial models be put together? Alternatively, consider combining behavioural models of component instances of the same type. Typically, several instances of the same component may appear in a given scenario, e.g., several instances of a client component that concurrently access a server. Standard approaches to synthesis produce a separate behavioural model for each client instance (e.g. [51, 35]). However, it is reasonable to integrate all models of all client instances into a model for the client component type because all clients should share the same characteristics. How can these partial models be composed?

Note that composition of behavioural models is not a novel idea [42, 24]; however, its main focus has been on parallel composition, which describes how two different components work together. In the context of model elaboration, we are interested in composing two partial descriptions of the *same* component to obtain a more elaborate version of both original partial descriptions. We call this operation a *merge*.

In a general merging framework, different stakeholders provide models and their properties. If the models can be merged (i.e., they are consistent), then a merged model that preserves the stakeholders' properties is constructed. If the models are inconsistent, then some form of feedback that gives insight into why the merge failed should be returned. Chechik and Uchitel [49] have instantiated this framework for an adaptation of MTSs [37]. In particular, they argue that the core concept underlying model merging is that of a common observational refinement, and define consistency as the existence of such a model. They show that merging consistent models is a process that should result in a minimal common observational refinement, and discuss the role of the least common observational refinement and the greatest common abstraction in this process.

In this thesis, we address several questions related to merge: (1) What properties are preserved in a merge? (2) When can two systems be merged? (3) What feedback can be returned when models cannot be merged? (4) When is merge unique? (5) How can complex models be merged? In addition, we provide several algorithms that are related to these questions, and a case study that illustrates our results on a realistic example.

1.2 Contributions of this Thesis

We extend the work of Chechik and Uchitel [49], and present many novelties related to merging MTSs: mathematical characterizations of merging and consistency, algebraic properties of the merge operator, new and improved algorithms related to constructing merge, and a case study. The specific contributions are as follows.

Firstly, we define a 3-valued counterpart of the logic weak μ-calculus and prove that it characterizes property preservation in merge (Question (1)). As weak μ-calculus can be quite subtle to use, we also define a 3-valued counterpart to the logic FLTL, intended for use as the main specification language in the merging framework, and prove that it is preserved by observational refinement (Question (1)).

Secondly, we provide a characterization of consistency using weak μ-calculus properties that distinguish between two systems, i.e., that evaluate to *true* in one system and *false* in the other. Such properties naturally characterize consistency for models with the same vocabulary. For models with different communicating alphabets, distinguishing properties are only meaningful when formulated and evaluated over the shared alphabet between the models. Consequently, deciding consistency in such contexts is non-trivial, and we therefore give sufficient conditions to prove soundness and completeness of a consistency characterization for this case (Questions (2) and (3)).

Thirdly, we introduce consistency relations defined between two MTSs and prove that there is a consistency relation between two systems if and only if they are consistent, which is another characterization of consistency (Question (2)). Using consistency relations relations, we give conditions for the uniqueness of merge, which is particularly relevant when attempting automation: if more than one merge exists, then some form of human intervention is required. We conclude that these conditions are consequences of the modelling notation (Question (4)).

Fourthly, We study algebraic properties of the merge operation to facilitate the merge of complex systems. We show that most desired properties hold when there is a unique merge, and give counterexamples and insights into why the same properties fail to hold when incomparable merges exist. However, we prove that the right choice of merge can usually be made with respect to a given algebraic property (Question (5)).

Fifthly, we give several algorithms related to merge, which no longer rely on the determinacy condition as a sufficient condition [49]. Specifically, we give improved algorithms for: (i) checking consistency, (ii) building a common refinement, and (iii) building a common abstraction that can elaborated into a minimal common observational refinement. We also give a new algorithm for constructing a weak μ-calculus property that distinguishes between inconsistent systems.

Lastly, we provide a case study that illustrates our results on a realistic example.

1.3 Organization of this Thesis

The rest of the thesis is organized as follows. In Chapter 2, we give preliminary definitions used throughout the thesis. Chapter 3 reviews merging MTSs, and is based on [49]. In Chapter 4, we define consistency relations, give sufficient conditions for merge to be unique, prove that observational refinement is characterized by 3-valued weak μ-calculus and preserves 3-valued FLTL, and use these results to prove a characterization of consistency. In Chapter 5, we present positive and negative results on algebraic properties for merging, both for the case in which merging yields a least common refinement and for the case in which it yields one of several different minimal common refinements, while providing insights on the implications of these results with respect to engineering partial behavioural models. In Chapter 6, we give several algorithms related to merge. In Chapter 7, we present a case study that illustrates the utility of our theoretical results for engineering and reasoning about partial descriptions of system behaviour. Finally, Chapter 8 presents a summary of our results, compares this work with related approaches, and discusses directions for future work.

Chapter 2

Background

In this chapter, we give and exemplify preliminary definitions and fix the notation. Section 2.1 discusses labelled transition systems, and modal transition systems that extend them. Section 2.2 reviews operations on MTSs: refinement, observation graphs, and parallel composition. Finally, Section 2.3 defines temporal logics weak μ-calculus and FLTL that will be used throughout the thesis to reason about MTSs.

2.1 Transitions Systems

We begin with the familiar concept of labelled transition systems (LTSs) [31], which are widely used for modelling and analyzing the behaviour of concurrent and distributed systems. An LTS is a state transition system where transitions are labelled with actions. The set of actions of an LTS is called its *communicating alphabet* and constitutes the interactions that the modelled system can have with its environment. In addition, LTSs can have transitions labelled with τ, representing actions that are not observable by the environment. Examples of a graphical representation of LTSs are models $\mathcal{A}$ and $\mathcal{B}$, given in Figure 2.1(a). In this thesis, the state labelled 0 is assumed to be the initial state of the transition system, unless stated otherwise. In addition, transitions labelled with sets are abbreviations for an individual transition on every action in the set, and the fonts $\mathcal{M}$, $\mathbb{M}$, and M are used for naming specific transition systems.

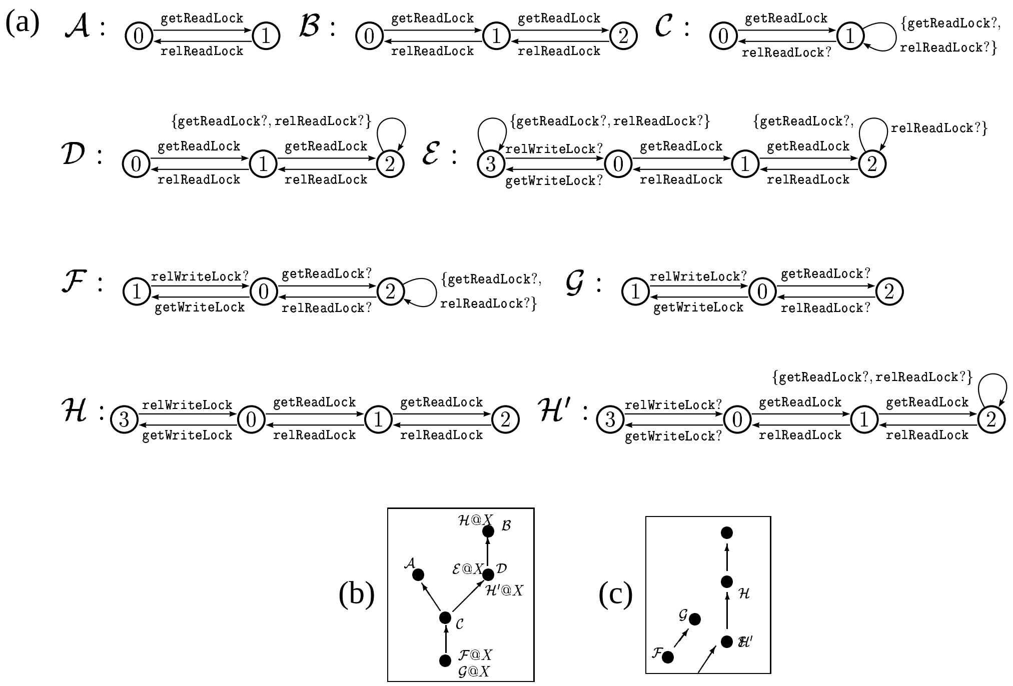

Figure 2.1: (a) LTSs and MTSs for reader and writer policies; Observational refinements: (b) over the alphabet $X = \{\texttt{getReadLock}, \texttt{relReadLock}\}$; (c) over the alphabet $X \cup \{\texttt{getWriteLock}, \texttt{relWriteLock}\}$.

Definition 1. (Labelled Transition Systems) *Let* States *be a universal set of states, Act be a universal set of observable action labels, and let $Act_\tau = Act \cup \{\tau\}$. A* Labelled Transition System *(LTS) is a tuple $P = (S, L, \Delta, s_0)$, where $S \subseteq$* States *is a finite set of states, $L \subseteq Act_\tau$ is a set of labels, $\Delta \subseteq (S \times L \times S)$ is a transition relation between states, and $s_0 \in S$ is the initial state. We use $\alpha P = L \setminus \{\tau\}$ to denote the communicating alphabet (vocabulary) of P.*

Existing semantics for LTSs assume that an LTS gives a complete behavioural description with respect to its alphabet. Consider LTS $\mathcal{A}$ which models a read lock. Starting in state 0, this model allows sequences of alternating `getReadLock` and `relReadLock` actions, and, by the completeness assumption, does not allow two `getReadLock` actions without having a `relReadLock` action in between them. LTS $\mathcal{A}$ is modelling a lock that can be held by at most one reader at any time. Model $\mathcal{B}$, on the other hand, allows two readers to hold the lock simultaneously. $\mathcal{A}$ and $\mathcal{B}$ are not considered to be equivalent under any of the standard equivalence relations such as strong bisimulation, trace, observational, or failure equivalence [24, 42].

Modal Transition Systems (MTSs) [37] allow explicit modelling of what is *not* known about the behaviour of a system. They extend LTSs with an additional set of transitions that model the interactions with the environment that the system cannot be guaranteed to provide, but equally cannot be guaranteed to prohibit.

Definition 2. (Modal Transition Systems) *A* Modal Transition System *(MTS) M is a structure $(S, L, \Delta^r, \Delta^p, s_0)$, where $\Delta^r \subseteq \Delta^p$, (S, L, Δ^r, s_0) is an LTS representing* required *transitions of the system and (S, L, Δ^p, s_0) is an LTS representing* possible *(but not necessarily required) transitions of the system. We use $\alpha M = L \setminus \{\tau\}$ to denote the communicating alphabet (vocabulary) of M.*

Figure 2.1(a) shows a graphical representation of some MTSs. For example, MTS $\mathcal{C}$ models a partial policy for a read lock that can be acquired by at least one reader at any time, but does not rule out concurrent readers. Transition labels that have a question mark are those in $\Delta^p \setminus \Delta^r$. We refer to these as “maybe” transitions, to distinguish them from required transitions (i.e., those in Δ^r). Note that LTSs are a special type of MTSs that do not have maybe transitions; thus, models $\mathcal{A}$ and $\mathcal{B}$ can be considered MTSs as well.

Given an MTS $M = (S, L, \Delta^r, \Delta^p, s_0)$, we say M transitions on ℓ through a required transition (denoted $M \xrightarrow{\ell}_r M'$) if $M' = (S, L, \Delta^r, \Delta^p, s_0')$ and $(s_0, \ell, s_0') \in \Delta^r$. Similarly, we say M transitions on ℓ through a maybe transition (denoted $M \xrightarrow{\ell}_m M'$) if $(s_0, \ell, s_0') \in \Delta^p \setminus \Delta^r$, and M transitions on ℓ through a possible transition (denoted $M \xrightarrow{\ell}_p M'$) if $M \xrightarrow{\ell}_r M'$ or $M \xrightarrow{\ell}_m M'$ (i.e., $(s_0, \ell, s_0') \in \Delta^p$). We write $M \xrightarrow{\ell}_p$ to mean $\exists M' \cdot M \xrightarrow{\ell}_p M'$. We say that M proscribes ℓ (denoted $M \not\xrightarrow{\ell}$) if M cannot transit on ℓ through maybe or required transitions. Finally, for an MTS $M = (S, L, \Delta^r, \Delta^p, s_0)$ and a state $n \in S$, we denote changing the initial state of M from s_0 to n as M_n. For example, some transitions of the MTS $\mathcal{C}$, shown in Figure 2.1(a), are $\mathcal{C}_0 \xrightarrow{\texttt{getReadLock}}_r \mathcal{C}_1$ (between states 0 and 1), and $\mathcal{C}_1 \xrightarrow{\texttt{relReadLock}}_m \mathcal{C}_1$ (a self-loop in state 1). Additionally, we call the set $\alpha M \cap \alpha N \cup \{\tau\}$ the *shared* actions between M and N, and $(\alpha M \setminus \alpha N) \cup (\alpha N \setminus \alpha M)$ the *non-shared* actions between M and N.

In this presentation, we associate each MTS M with its communicating alphabet αM, extending the presentation of [37]. The communicating alphabet is the set of events that are relevant to the model, i.e., the scope of a partial description. Allowing models to have different scopes is fundamental for merging descriptions of different concerns and viewpoints, as discussed in Chapter 3. In addition, our choice is in line with process algebra semantics such as FSP [41].

2.2 Refinement

Refinement of MTSs captures the notion of elaboration of a partial description into a more comprehensive one, in which some knowledge over the maybe behaviour has been gained. It can be seen as being a "more defined than" relation between two partial models. Intuitively, refinement in MTSs is about converting maybe transitions into required transitions or removing them altogether: an MTS N refines an MTS M if N preserves all of the required and all of the proscribed behaviours of M. Alternatively, N refines M if N can simulate the required behaviour of M, and M can simulate the possible behaviour of N.

Definition 3. (Refinement) *Let $\wp$ be the universe of all MTSs. N is a refinement of M, written $M \preceq N$, when $\alpha M = \alpha N$ and (M, N) is contained in some* refinement relation $R \subseteq \wp \times \wp$ *for which the following holds for all $\ell \in Act_\tau$ and all $(M, N) \in R$:*

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

We often refer to this form of refinement as *strong* refinement.

Consider the MTSs shown in Figure 2.1(a). MTS $\mathcal{C}$ is refined by the LTS $\mathcal{A}$ ($\mathcal{C} \preceq \mathcal{A}$), incorporating the new knowledge that the maybe self-loop at state $\mathcal{C}_1$ should be removed. The refinement relation between these models is $R = \{(\mathcal{C}_0, \mathcal{A}_0), (\mathcal{C}_1, \mathcal{A}_1)\}$. The LTS $\mathcal{B}$ refines $\mathcal{C}$ ($\mathcal{C} \preceq \mathcal{B}$), with the refinement relation $R = \{(\mathcal{C}_0, \mathcal{B}_0), (\mathcal{C}_1, \mathcal{B}_1), (\mathcal{C}_1, \mathcal{B}_2)\}$. Finally, the MTS $\mathcal{D}$ refines $\mathcal{C}$ via the relation $R = \{(\mathcal{C}_0, \mathcal{D}_0), (\mathcal{C}_1, \mathcal{D}_1), (\mathcal{C}_1, \mathcal{D}_2)\}$.

$$\frac{M \xrightarrow{\ell}_{\gamma} M'}{(M\backslash X) \xrightarrow{\ell}_{\gamma} (M'\backslash X)}\ \ell \notin X \qquad \frac{M \xrightarrow{\ell}_{\gamma} M'}{(M\backslash X) \xrightarrow{\tau}_{\gamma} (M'\backslash X)}\ \ell \in X$$

Figure 2.2: Rules for the hiding operator.

Although refinement captures the notion of model elaboration, it requires the alphabets of the processes being compared to be equal. In practice, model elaboration can lead to augmenting the alphabet of the system model to describe behavioural aspects that previously had not been taken into account. To capture this aspect of model elaboration, we introduce two concepts: hiding and observational refinement.

Hiding is an operation that makes a set of actions of a model unobservable to its environment by reducing the alphabet of the model and replacing transitions labelled with an action in the hiding set by τ, as shown in Figure 2.2.

Definition 4. (Hiding) *Let* $M = (S, L, \Delta^r, \Delta^p, s_0)$ *be an MTS and* $X \subseteq Act$ *be a set of observable actions.* M *with the actions of* X *hidden, denoted* $M\backslash X$*, is an MTS* $(S, L\backslash X, \Delta^{r'}, \Delta^{p'}, s_0)$*, where* $\Delta^{r'}$ *and* $\Delta^{p'}$ *are the smallest relations that satisfy the rules in Figure 2.2, where* $\gamma \in \{r, m\}$*. We use* $M@X$ *to denote* $M\backslash(Act\backslash X)$*.*

Let $w = w_1, \ldots, w_k$ be a word over Act_τ. $M \xrightarrow{w}_r M'$ means that there exist $M_0, \ldots, M_k$ such that $M_0 = M$, $M_k = M'$, and $M_i \xrightarrow{w_{i+1}}_r M_{i+1}$ for $0 \leq i < k$. $M \xrightarrow{w}_m M'$ means that there exist $M_0, \ldots, M_k$ such that $M_0 = M$, $M_k = M'$, $M_i \xrightarrow{w_{i+1}}_p M_{i+1}$ for $0 \leq i < k$, and $\exists j \cdot 0 \leq j \leq k \cdot M_j \xrightarrow{w_{j+1}}_m M_{j+1}$, i.e., there is at least one maybe transition on some letter of w. For $\gamma \in \{r, p\}$, we write $M \overset{\epsilon}{\Longrightarrow}_\gamma M'$ to denote $M(\xrightarrow{\tau}_\gamma)^* M'$, and $M \overset{\epsilon}{\Longrightarrow}_m M'$ to denote $M(\overset{\epsilon}{\Longrightarrow}_p)(\xrightarrow{\tau}_m)(\overset{\epsilon}{\Longrightarrow}_p)M'$, i.e., there is at least one maybe transition on τ. For $\ell \neq \tau$ and $\gamma \in \{r, p\}$, we write $M \overset{\ell}{\Longrightarrow}_\gamma M'$ to denote $M(\overset{\epsilon}{\Longrightarrow}_\gamma)(\xrightarrow{\ell}_\gamma)(\overset{\epsilon}{\Longrightarrow}_\gamma)M'$, and $M \overset{\ell}{\Longrightarrow}_m M'$ for $M(\overset{\epsilon}{\Longrightarrow}_m)(\xrightarrow{\ell}_p)(\overset{\epsilon}{\Longrightarrow}_p)M'$ or $M(\overset{\epsilon}{\Longrightarrow}_p)(\xrightarrow{\ell}_m)(\overset{\epsilon}{\Longrightarrow}_p)M'$, i.e., the maybe transition precedes or is on ℓ along the path from M to M'. For $\gamma \in \{r, m, p\}$, we extend $\Longrightarrow_\gamma$ to words in the same way as for $\longrightarrow_\gamma$. For $\ell \in Act_\tau$, let $\hat{\ell} = \ell$ if $\ell \neq \tau$ and $\hat{\ell} = \epsilon$ if $\ell = \tau$. For $\gamma \in \{r, m, p\}$ and $\ell \in Act_\tau$, we often write $s \xrightarrow{\ell}_\gamma s'$ to mean $M_s \xrightarrow{\ell}_\gamma M_{s'}$ and similarly for $\Longrightarrow_\gamma$. Transitions on the thin arrow $\longrightarrow_\gamma$ are referred to as *simple* transitions, and transitions on the thick arrow

$\Longrightarrow_\gamma$ are referred to as *observable* transitions.

To compare a model with another one with an augmented alphabet, we must hide the additional actions in the second model and then use *observational refinement* – effectively, refinement that ignores differences in the precise amount of τ transitions.

Definition 5. (Observational Refinement) *N is an observational refinement of M, written $M \preceq_o N$, if $\alpha M = \alpha N$ and (M, N) is contained in some refinement relation $R \subseteq \wp \times \wp$ for which the following holds for all $\ell \in Act_\tau$ and all $(M, N) \in R$:*

$$1. \quad (M \xrightarrow{\ell}_r M') \Rightarrow (\exists N' \cdot N \overset{\hat{\ell}}{\Longrightarrow}_r N' \ \wedge\ (M', N') \in R)$$
$$2. \quad (N \xrightarrow{\ell}_p N') \Rightarrow (\exists M' \cdot M \overset{\hat{\ell}}{\Longrightarrow}_p M' \ \wedge\ (M', N') \in R)$$

Consider again the MTSs shown in Figure 2.1(a). For model $\mathcal{E}$, if a process acquires the write lock, then the read lock cannot be acquired until the write lock is released. Note that this model does not indicate that processes are actually allowed to acquire the write lock, as these transitions are maybe. Hiding the actions `getWriteLock` and `relWriteLock` results in an MTS just like $\mathcal{E}$, but with labels `getWriteLock`? and `relWriteLock`? changed to τ?. Furthermore, the resulting model is observationally refined by the MTS $\mathcal{D}$:

$$\mathcal{E}' = \mathcal{E} \setminus \{\texttt{getWriteLock}, \texttt{relWriteLock}\} \preceq_o \mathcal{D},$$

where the refinement relation is:

$$R = \{(\mathcal{E}'_0, \mathcal{D}_0), (\mathcal{E}'_1, \mathcal{D}_1), (\mathcal{E}'_2, \mathcal{D}_2), (\mathcal{E}'_3, \mathcal{D}_0)\}.$$

In fact, $\mathcal{E}'$ is also a refinement of $\mathcal{D}$ via the inverse of R. We say that $\mathcal{E}'$ and $\mathcal{D}$ are *observationally equivalent*, written $\mathcal{E}' \equiv_o \mathcal{D}$.

Figures 2.1(b) and 2.1(c) depict observational refinements that hold between models in Figure 2.1(a). Each graph relates models with the same alphabet:

$$X = \{\texttt{getReadLock}, \texttt{relReadLock}\}$$

in Figure 2.1(b), and

$$Y = X \cup \{\texttt{getWriteLock}, \texttt{relWriteLock}\}$$

in Figure 2.1(c). Nodes with multiple labels indicate models that are observationally equivalent. Note that models $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ have the alphabet X. Consequently, they cannot be related through observational refinement to models with an augmented alphabet Y, i.e. $\mathcal{E}$, $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{H}'$. For this reason, $\mathcal{A}$ through $\mathcal{D}$ do not appear in Figure 2.1(c). However, these models can be related through observational refinement to $\mathcal{E}$, $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{H}'$ if the latter have their alphabets restricted to X, and are depicted in Figure 2.1(b).

It is often desirable to abstract a model from transitions resulting from internal computation. One way of doing this for MTSs is to use *observation graphs* [38], as is traditionally done for LTSs [9]. Intuitively, observation graphs absorb the τ transitions of a model into observable actions, so that the exact amount of internal behavior is obscured.

Definition 6. (Observation Graph) *Let* $Act_\epsilon = Act \cup \{\epsilon\}$*. For an MTS* $M = (S, L, \Delta^r, \Delta^p, s_0)$*, the* observation graph *of* M *is the derived MTS* $M_\epsilon = (S,\, L \cup \{\epsilon\},\, \Delta^r_\epsilon,\, \Delta^p_\epsilon,\, s_0)$*, where* Δ^r_ϵ *and* Δ^p_ϵ *are defined as follows for all* $\ell \in Act_\epsilon$*:*

$$1.\quad M_\epsilon \xrightarrow{\ell}_r M'_\epsilon \Leftrightarrow M \overset{\ell}{\Longrightarrow}_r M'$$
$$2.\quad M_\epsilon \xrightarrow{\ell}_m M'_\epsilon \Leftrightarrow M \overset{\ell}{\Longrightarrow}_m M'$$

In particular, absorption of τ transitions into visible actions is done by defining the relation $\xrightarrow{\epsilon}_r$ in M_ϵ as the reflexive and transitive closure of $\xrightarrow{\tau}_r$ in M, and $\xrightarrow{\epsilon}_m$ as the transitive closure of $\xrightarrow{\tau}_m$ in M. Transitions on $\ell \neq \epsilon$ are defined by the relational products of $\overset{\epsilon}{\Longrightarrow}$ and $\xrightarrow{\ell}$ in M. In fact, it follows that $M \preceq_o N$ if and only if $M_\epsilon \preceq N_\epsilon$. That is, observational refinement is intuitively about gaining some knowledge over the maybe behaviour in the observation graph of the original system, i.e., maybe transitions either stay maybe, or are refined to true or false in the more refined model.

Larsen and Thomsen [37] define a parallel composition operator over MTSs, intended to describe how models of two different systems work together.

Definition 7. (Parallel Composition) *Let* $M = (S_M,\, L_M,\, \Delta^r_M,\, \Delta^p_M,\, s_{0M})$ *and* $N = (S_N,\, L_N,\, \Delta^r_N,\, \Delta^p_N,\, s_{0N})$ *be MTSs.* Parallel composition *(*$\|$*) is a symmetric operator such that* $M \| P$ *is*

$$\text{TD}\ \frac{M \xrightarrow{\ell}_r M'}{M||N \xrightarrow{\ell}_r M'||N}\ \ell \notin \alpha N \qquad \text{MT}\ \frac{M \xrightarrow{\ell}_m M',\ N \xrightarrow{\ell}_r N'}{M||N \xrightarrow{\ell}_m M'||N'}\ \ell \neq \tau \qquad \text{MD}\ \frac{M \xrightarrow{\ell}_m M'}{M||N \xrightarrow{\ell}_m M'||N}\ \ell \notin \alpha N$$

$$\text{TT}\ \frac{M \xrightarrow{\ell}_r M',\ N \xrightarrow{\ell}_r N'}{M||N \xrightarrow{\ell}_r M'||N'}\ \ell \neq \tau \qquad \text{MM}\ \frac{M \xrightarrow{\ell}_m M',\ N \xrightarrow{\ell}_m N'}{M||N \xrightarrow{\ell}_m M'||N'}\ \ell \neq \tau$$

Figure 2.3: Rules for parallel composition.

the MTS $(S_M \times S_N, L_M \cup L_N, \Delta^r, \Delta^p, (s_{0M}, s_{0N}))$*, where* Δ^r *and* Δ^p *are the smallest relations that satisfy the rules given in Figure 2.3.*

For example, in Figure 2.1(a), $\mathcal{A}||\mathcal{G} = \mathcal{G}$, assuming that $\{\texttt{getWriteLock}, \texttt{relWriteLock}\} \cap \alpha\mathcal{A} = \emptyset$. Note that in the rules in Figure 2.3, "T" stands for "true", "M" stands for "maybe", and "D" stands for "don't care". In particular, rule TT captures the case when there is a true (required) transition in both models, MT captures the case when there is a maybe transition in one model and a true (required) transition in the other, and TD captures the case when there is a required transition in one model on a non-shared action (i.e., on an action the other system is not concerned with).

Proposition 1. (Properties of || [37]) *Parallel composition satisfies the following properties:*

1. *(Commutativity)* $M||N = N||M$.
2. *(Associativity)* $(M||N)||P = M||(N||P)$.
3. *(Monotonicity)* $M \preceq_o N \Rightarrow M||P \preceq_o N||P$.

2.3 Temporal Logics

In this section, we review the Kleene logic, and use it to define 3-valued counterparts to the temporal logics weak μ-calculus and Fluent Linear Temporal Logic (FLTL), used for reasoning about MTSs.

2.3.1 Kleene Logic

The truth values $\mathbf{t}$ (*true*), $\mathbf{f}$ (*false*), and $\bot$ (*maybe*) form the Kleene logic [32], which we refer to as $\mathbf{3}$. The values $\mathbf{t}$ and $\mathbf{f}$ behave classically with respect to $\wedge$ (and), $\vee$ (or), and $\neg$ (negation).

The following identities hold for $\bot$:

$$\bot \wedge \mathbf{t} = \bot \qquad \bot \wedge \mathbf{f} = \mathbf{f} \qquad \bot \vee \mathbf{t} = \mathbf{t} \qquad \bot \vee \mathbf{f} = \bot \qquad \neg\bot = \bot.$$

This logic has two useful orderings, $\leq$ (truth) and $\leq_{\mathit{inf}}$ (information), which satisfy $\mathbf{f} \leq \bot \leq \mathbf{t}$, and $\bot \leq_{\mathit{inf}} \mathbf{t}$ and $\bot \leq_{\mathit{inf}} \mathbf{f}$. That is, *maybe* gives the least amount of information: it could be either *true* or *false* .

2.3.2 Weak μ-calculus

While shown in [29] to characterize strong refinement, the 3-valued counterpart to μ-calculus ($\mathcal{L}_\mu$) [33] is not well-suited for describing the observable behaviour of an MTS and does not characterize *observational* refinement, because it makes no distinction between an observable action and τ. Instead, we define a 3-valued extension of *weak μ-calculus* ($\mathcal{L}_\mu^w$), which does make such a distinction. The 2-valued version of this logic has been shown to be a useful logic for expressing properties of LTSs in [48].

3-valued $\mathcal{L}_\mu^w$ enables a formula to evaluate to an element of $\mathbf{3}$. For a set of fixed point variables *Var*, $a \in Act_\tau$ and $Z \in \mathit{Var}$, an $\mathcal{L}_\mu^w$ formula ϕ has the grammar:

$$\phi \triangleq \mathbf{t} \mid \mathbf{f} \mid \bot \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle a \rangle_o \phi \mid [a]_o \phi \mid \mu Z.\phi \mid \nu Z.\phi,$$

where $\langle a \rangle_o$ and $[a]_o$ are the *next* operators with intended meanings “exists a next state reachable via an observable transition on a” and “for all next states reachable via an observable transition on a”, respectively. We write $\phi(Z)$ to denote a formula that might contain free occurrences of the variable Z. μ and ν represent the least and greatest fixed points, respectively, and we write $\langle \cdot \rangle_o \phi$ and $[\cdot]_o \phi$ as abbreviations for $\exists a \in Act \cdot \langle a \rangle_o \phi$ and $\forall a \in Act \cdot [a]_o \phi$.

Let ϕ be a formula in $\mathcal{L}_\mu^w$, $M = (S_M, L_M, \Delta_M^r, \Delta_M^p, s_0)$ be an MTS, and $e_1, e_2 : \mathit{Var} \to \mathcal{P}(S_M)$ be *environments* mapping fixed point variables to sets of states. $[\![\phi]\!]_{e_1}^{\mathbf{t}}$ ($[\![\phi]\!]_{e_2}^{\mathbf{f}}$) denotes the set of states in M where ϕ is *true* (*false*). The set of states where ϕ is *maybe* is then $S_M \backslash ([\![\phi]\!]_{e_1}^{\mathbf{t}} \cup [\![\phi]\!]_{e_2}^{\mathbf{f}})$ (i.e., ϕ is not *true* or *false*).

$$\begin{array}{rclrcl}
\llbracket \mathbf{t} \rrbracket^{\mathbf{t}}_{e_1} & \triangleq & S_M & \llbracket \varphi \wedge \psi \rrbracket^{\mathbf{t}}_{e_1} & \triangleq & \llbracket \varphi \rrbracket^{\mathbf{t}}_{e_1} \cap \llbracket \psi \rrbracket^{\mathbf{t}}_{e_1} \\
\llbracket \mathbf{t} \rrbracket^{\mathbf{f}}_{e_2} & \triangleq & \emptyset & \llbracket \varphi \wedge \psi \rrbracket^{\mathbf{f}}_{e_2} & \triangleq & \llbracket \varphi \rrbracket^{\mathbf{f}}_{e_2} \cup \llbracket \psi \rrbracket^{\mathbf{f}}_{e_2} \\
\llbracket \bot \rrbracket^{\mathbf{t}}_{e_1} & \triangleq & \emptyset & \llbracket \langle a \rangle_o \phi \rrbracket^{\mathbf{t}}_{e_1} & \triangleq & \{ s \in S_M \mid \exists s' \in S_M \cdot (s \xRightarrow{a}_r s' \wedge s' \in \llbracket \phi \rrbracket^{\mathbf{t}}_{e_1}) \} \\
\llbracket Z \rrbracket^{\mathbf{t}}_{e_1} & \triangleq & e_1(Z) & \llbracket \langle a \rangle_o \phi \rrbracket^{\mathbf{f}}_{e_2} & \triangleq & \{ s \in S_M \mid \forall s' \in S_M \cdot (s \xRightarrow{a}_p s' \Rightarrow s' \in \llbracket \phi \rrbracket^{\mathbf{f}}_{e_2}) \} \\
\llbracket Z \rrbracket^{\mathbf{f}}_{e_2} & \triangleq & e_2(Z) & \llbracket \mu Z.\phi(Z) \rrbracket^{\mathbf{t}}_{e_1} & \triangleq & \cap \{ S \subseteq S_M \mid \llbracket \phi \rrbracket^{\mathbf{t}}_{e_1[Z \rightarrow S]} \subseteq S \} \\
\llbracket \neg \phi \rrbracket^{\mathbf{t}}_{e_1} & \triangleq & \llbracket \phi \rrbracket^{\mathbf{f}}_{e_2} & \llbracket \mu Z.\phi(Z) \rrbracket^{\mathbf{f}}_{e_2} & \triangleq & \cap \{ S \subseteq S_M \mid \llbracket \phi \rrbracket^{\mathbf{f}}_{e_2[Z \rightarrow S]} \subseteq S \}
\end{array}$$

Figure 2.4: 3-valued semantics of $\mathcal{L}^w_\mu$.

Definition 8. (3-valued Semantics of $\mathcal{L}^w_\mu$) *For an MTS M, a formula ϕ in $\mathcal{L}^w_\mu$, and environments e_1 and e_2, $\llbracket \phi \rrbracket^{\mathbf{t}}_{e_1} \subseteq S_M$ and $\llbracket \phi \rrbracket^{\mathbf{f}}_{e_2} \subseteq S_M$ are defined as shown in Figure 2.4, where $a \in \mathrm{Act}_\epsilon$, and $e_i[Z \rightarrow S]$ is the same environment as e_i except it maps Z to S.*

$\phi_1 \vee \phi_2$, $[a]_o\phi$ and $\nu Z.\phi(Z)$ are defined through negation: $\phi_1 \vee \phi_2 = \neg\phi_1 \wedge \neg\phi_2$, $[a]_o\phi = \neg\langle a \rangle_o \neg\phi$, and $\nu Z.\phi(Z) = \neg\mu Z.\phi(\neg Z)$. The value of ϕ in M is its value in the initial state. We omit the environments from $\llbracket \phi \rrbracket^{\mathbf{t}}$ and $\llbracket \phi \rrbracket^{\mathbf{f}}$ to mean that e_1 and e_2 map every Z in *Var* to $\emptyset$ and S_M, respectively. Finally, if M is an LTS, the semantics in Definition 8 reduces to the standard 2-valued semantics in [40].

For example, the property $\langle a \rangle_o \mathbf{t}$ (which expresses the ability to perform an *observable* transition on a) evaluates to *true* in both I and J in Figure 2.5, even though the transition on a in J is preceded by a τ. In particular, $\langle a \rangle_o$ and $[a]_o$ allow zero or more τ transitions before and after the observable action a, but the precise amount of internal computation is unimportant. Additionally, the property $[a]_o\langle b \rangle_o\mathbf{t}$ is *maybe* in I because $\mathsf{I}_0 \xrightarrow{a}_r \mathsf{I}_1$ is the only transition on a from the initial state and $\mathsf{I}_1 \xrightarrow{b}_m \mathsf{I}_1$ is the only transition on b from I_1. Finally, $[\epsilon]_o\langle a \rangle_o\mathbf{f}$ is *false* in K because $\mathsf{K}_0 \xRightarrow{\epsilon}_r \mathsf{K}_0$ and $0 \in \llbracket \langle a \rangle_o \mathbf{f} \rrbracket^{\mathbf{f}}$. That is, $\mathcal{L}^w_\mu$ properties that explicitly involve internal activity always allow for the possibility of staying in the same state, even if no self-loops on τ are present.

2-valued $\mathcal{L}^w_\mu$ is a fragment of $\mathcal{L}_\mu$ [40]. For the 3-valued counterparts of these logics, this means that for every formula ϕ_w in $\mathcal{L}^w_\mu$ there is a formula ϕ_s in $\mathcal{L}_\mu$ such that $\llbracket \phi_w \rrbracket^{\mathbf{t}} = \llbracket \phi_s \rrbracket^{\mathbf{t}}$

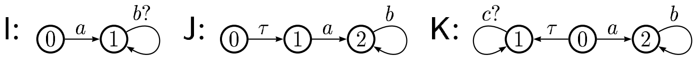

Figure 2.5: MTSs for illustrating temporal properties.

and $[\![\phi_w]\!]^{\mathbf{f}} = [\![\phi_s]\!]^{\mathbf{f}}$. The translation procedure follows from the fact that internal activity can be expressed through extra fixed point computations in $\mathcal{L}_\mu$.

Additionally, $\mathcal{L}_\mu^w$ subsumes the expressive power of temporal logics CTL$_{-X}$ and LTL$_{-X}$ (i.e., CTL and LTL [7] without the next operators), and these relationships hold for the 3-valued versions of the logics as well.

2.3.3 FLTL

Although $\mathcal{L}_\mu^w$ is very expressive and in fact characterizes merging MTSs (see Chapter 4), it can be quite subtle and difficult to use. We therefore define a 3-valued version of FLTL [16], which is a simple yet expressive logic for capturing properties that combine state and action, intended for use as our primary specification language.

A fluent *Fl* is defined by a pair of sets I_{Fl}, the set of initiating actions, and T_{Fl}, the set of terminating actions:

$$Fl = \langle I_{Fl}, T_{Fl} \rangle \text{ where } I_{Fl}, T_{Fl} \subseteq Act \text{ and } I_{Fl} \cap T_{Fl} = \emptyset.$$

A fluent may be initially *true* or *false* as indicated by the *Initially*$_{Fl}$ attribute, where a lack of this attribute indicates that the fluent is initially *false*. Every action $a \in Act$ induces a fluent, namely $a = \langle a, Act \setminus \{a\} \rangle$.

Given the set of fluents Φ, a well-defined FLTL formula is defined inductively using the standard boolean connectives, **X** (next), **U** (strong until), **F** (eventually), and **G** (always), as follows:

$$Fl \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \mathbf{X}\phi \mid \phi\mathbf{U}\psi \mid \mathbf{F}\phi \mid \mathbf{G}\phi,$$

where $Fl \in \Phi$. For an infinite trace $\pi = a_0, a_1, a_2, \ldots$, over *Act* and some $i \in \mathbb{N}$, we write π^i for the suffix of π starting at a_i. Let Π be the set of *infinite* traces over *Act*. An MTS M induces a

$$\begin{array}{cc}
\pi \models \mathit{Fl} \triangleq \pi^0 \models \mathit{Fl} & \pi \models \phi \wedge \psi \triangleq (\pi \models \phi) \wedge (\pi \models \psi) \\
\pi \models \neg\phi \triangleq \neg(\pi \models \phi) & \pi \models \mathbf{X}\phi \triangleq \pi^1 \models \phi \\
\pi \models \phi \vee \psi \triangleq (\pi \models \phi) \vee (\pi \models \psi) & \pi \models \phi \mathbf{U} \psi \triangleq \exists i \geq 0 \cdot \pi^i \models \psi \wedge \forall\, 0 \leq j < i \cdot \pi^j \models \phi
\end{array}$$

Figure 2.6: Semantics for the satisfaction operator.

function $M : \Pi \rightarrow \mathbf{3}$ that assigns values to traces.

Definition 9. (Value of a Trace) *A trace π in Π is a* true trace *in M (i.e., $M(\pi) = \mathbf{t}$) if there exists an infinite sequence $\{M_i\}$ such that $M_0 = M$ and $M_i \overset{a_i}{\Longrightarrow}_r M_{i+1}$ for all $i \in \mathbb{N}$. A trace π is a* maybe trace *in M (i.e., $M(\pi) = \bot$) if π is not a true trace, but there exists an infinite sequence $\{M_i\}$ such that $M_0 = M$, $M_i \overset{a_i}{\Longrightarrow}_p M_{i+1}$ for all $i \in \mathbb{N}$, and $M_j \overset{a_j}{\Longrightarrow}_m M_{j+1}$ for at least one $j \in \mathbb{N}$. A trace π is a* possible trace *in M (i.e., $M(\pi) \geq \bot$) if π is a maybe or true trace in M. Finally, a trace π is a* false trace *in M (i.e., $M(\pi) = \mathbf{f}$) if it is not a possible trace.*

Given a trace $\pi \in \Pi$, a suffix π^i satisfies a fluent *Fl*, denoted $\pi \models \mathit{Fl}$, if and only if one of the following conditions holds:

- $(\mathit{Initially}_{\mathit{Fl}} \wedge (\forall j \in \mathbb{N} \cdot 0 \leq j \leq i \Rightarrow a_j \notin T_{\mathit{Fl}}))$
- $(\exists j \in \mathbb{N} \cdot (j \leq i \wedge a_j \in I_f) \wedge (\forall k \in \mathbb{N} \cdot j < k \leq i \Rightarrow a_k \notin T_{\mathit{Fl}})$

In other words, a fluent holds at a time instant if and only if it holds initially or some initiating action has occurred, and in both cases, no terminating action has yet occurred. Note that the interval over which a fluent holds is *closed* on the left and *open* on the right, since actions have an immediate effect on the value of fluents.

The 3-valued semantics of FLTL over an MTS M is given by the function $||\cdot||^M$ that, for each formula $\phi \in$ FLTL, returns the value of ϕ in M.

Definition 10. (3-valued Semantics of FLTL) *For an MTS M, the function $||\cdot||^M$:* FLTL $\longrightarrow \mathbf{3}$ *is defined as follows:*

$$||\phi||^M \triangleq \bigwedge_{\pi \in \Pi} (M(\pi) \Rightarrow (\pi \models \phi)),$$

where the semantics for $\pi \models \phi$ is given in Figure 2.6.

The operators $\mathbf{F}$ and $\mathbf{G}$ are defined as $\mathbf{F}\phi = \mathbf{t}\mathbf{U}\phi$ and $\mathbf{G} = \neg\mathbf{F}\neg\phi$.

Hence, M satisfies ϕ (i.e., $||\phi||^M = \mathbf{t}$) if every possible trace π in M satisfies ϕ, since $M(\pi) \geq \bot$ and $\pi \models \phi$ for any such π, making the implication $(M(\pi) \Rightarrow (\pi \models \phi))$ *true*. M refutes ϕ (i.e., $||\phi||^M = \mathbf{f}$) if there is at least one true trace π that does not satisfy ϕ, since $M(\pi) = \mathbf{t}$ and $\pi \not\models \phi$ for any such π, making the implication $(M(\pi) \Rightarrow (\pi \models \phi))$ *false*. Otherwise, the value of ϕ in M is *maybe* (i.e., $||\phi||^M = \bot$). Note that if π is a false trace in M, then $(M(\pi) \Rightarrow (\pi \models \phi))$ is necessarily true, regardless of whether π satisfies ϕ. Hence, we can ignore false traces in M when determining the value of $||\phi||^M$, as we would expect. Also, only infinite traces are considered when determining the value of an FLTL formula. If M is an LTS, the 3-valued semantics in Definition 10 reduces to the standard 2-valued semantics given in [16].

For examples of FLTL properties, refer to models I, J, and K in Figure 2.5 and consider the FLTL property $\phi_1 = \mathbf{F}b$ (eventually the fluent induced by action $\mathtt{b}$ is *true*). ϕ_1 is *true* in I and J (i.e, $||\phi_1||^{\mathsf{I}} = ||\phi_1||^{\mathsf{J}} = \mathbf{t}$) because the only possible trace in both models is $\pi_1 = \mathtt{a},\mathtt{b},\mathtt{b},\mathtt{b},\ldots$, and clearly $\pi_1 \models \phi_1$. ϕ_1 is *maybe* in K (i.e., $||\phi_1||^{\mathsf{K}} = \bot$) because K admits the additional maybe trace $\pi_2 = \mathtt{c},\mathtt{c},\mathtt{c},\ldots$, and $\pi_2 \not\models \phi_1$. The property $\phi_2 = \mathbf{G}b$ is *false* in I because $\mathtt{a},\mathtt{b},\mathtt{b},\mathtt{b},\ldots$, is a true trace in I that does not satisfy ϕ_2. Finally, the property $\neg b\mathbf{U}\neg a$, which says that a transition on $\mathtt{b}$ cannot be performed until a transition on $\mathtt{a}$ is not possible, is *true* in all three models. Note that the equivalent $\mathcal{L}_\mu^w$ property is $\mu Z.(\langle a\rangle_o\mathbf{f} \vee (\langle b\rangle_o\mathbf{f} \wedge \langle\cdot\rangle_o\mathbf{t} \wedge [\cdot]_o Z))$, which is much less intuitive.

2.3.4 Model-Checking

In the previous example, π_1 is a maybe trace in I and a true trace in J, yet both models satisfy $\mathbf{F}b$. If the maybe transition $\mathsf{I}_1 \xrightarrow{b}_m \mathsf{I}_1$ is refined to false, there are no possible traces in I, and therefore l satisfies every FLTL formula. On the other hand, if it is refined to true, then π_1 is a true trace that satisfies ϕ. Hence, l satisfies ϕ, regardless of the value of $\mathsf{l}_1 \xrightarrow{b}_m \mathsf{l}_1$.

We can use the above intuition to implement model-checking of our 3-valued temporal logics. For an MTS M, define M^+ to be the LTS obtained from M by converting all maybe transitions into required transitions, and M^- to be the LTS obtained from M by converting all maybe transitions into false transitions. The following theorem is adapted from [4].

Theorem 1. (3-valued Model-Checking via M^+ and M^-) *For an MTS* M *and* $\phi \in \mathcal{L}_\mu^w$ *or* $\phi \in$ FLTL*:*

1. *If* ϕ *is true in* M^+ *and* M^-*, it is true in* M*,*
2. *if* ϕ *is false in* M^+ *and* M^-*, it is false in* M*;*
3. *otherwise,* ϕ *is maybe in* M

Hence, model-checking of these logics can be done by two calls to a classical model-checker, e.g, 3-valued FLTL model-checking can be supported by the LTSA tool [41].

For example, consider $\phi = [a]_o\langle b\rangle_o\mathbf{t}$. If ϕ is *false* in M^-, then there exists M'^- such that $M^- \overset{a}{\Longrightarrow}_r M'^-$ and $M'^- \overset{b}{\not\Longrightarrow}$. Since true transitions in M^- correspond to true transitions in M, ϕ is also *false* in M. Now suppose that ϕ is *true* in M^-. There may exist M' such that $M \overset{a}{\Longrightarrow}_m M'$ and $M' \overset{b}{\not\Longrightarrow}$, which would not be taken into account in M^-. Therefore, we must also check if ϕ is *true* in M^+ to conclude that ϕ is *true* in M. Note that only checking if ϕ is *true* in M^+ is insufficient as well. It may be that case that whenever $M \overset{a}{\Longrightarrow}_p M'$, it is only true that $M' \overset{b}{\Longrightarrow}_m$, making ϕ *maybe* in M. Hence, the value of ϕ must be checked in both M^+ and M^- in order to conclude its value in M.

If ϕ is an FLTL formula, it is sufficient for ϕ to be *true* (*false*) in M^+ (M^-) in order to conclude that ϕ is *true* (*false*) in M, since true traces in M^+ and possible traces in M correspond, and true traces in M^- and true traces in M correspond. If ϕ is *false* in M^+ and *true* in M^-, then ϕ is *maybe* in M.

Chapter 3

Merging Models

In this chapter, we review merging modal transition systems. Section 3.1 defines the notion of a common observational refinement as the basis for merge. Section 3.2 defines the least common refinement as the merge, if it exists, and a minimal common refinement otherwise. Finally, Section 3.3 discusses the role of the greatest common abstraction in the case that the least common refinement does not exist, and we end with a summary in Section 3.4.

Figure 3.1 provides an abstract summary of the concepts discussed in this chapter. In this figure, arrows depict observational refinements (i.e., an edge from P to Q indicates that Q is a refinement of P). All models are assumed to be defined over the same alphabet, and transitive relations are not depicted.

3.1 Common Observational Refinement

The intuition we wish to capture by merging is that of augmenting the knowledge we have of the behaviour of a system by taking what we know from the two partial descriptions of the system. Clearly, the notion of refinement underlies this intuition as it captures the "more defined than" relation between two partial models. Hence, merging two models of the same system is about finding a common refinement for these models, i.e., finding a model that is more defined than both.

It is possible that the models being merged have different alphabets, and therefore merging these models results in a model whose alphabet is a superset of the original ones: both models are extended with the information that the other has. When comparing such a merge to one of the original systems, transitions on actions that are out of the scope of this system are effectively the same as internal activity (i.e., transitions on τ). In addition, the precise amount of such activity arising from hiding actions that are out of the scope of one system is irrelevant, as this behaviour is unobservable to the system in question. Therefore, when comparing a merge to one of the original systems, differences in internal activity occurring before and after observable behaviour should, in essence, be ignored. Hence, the merge of two partial behavioural models should be an *observational* refinement of each of the original systems, with the appropriate alphabet restrictions.

Definition 11. (Common Observational Refinement) *An MTS P is a* common refinement *(CR) of MTSs M and N if $\alpha P \supseteq (\alpha M \cup \alpha N)$, $M \preceq_o P@\alpha M$ and $N \preceq_o P@\alpha N$.*

We write $\mathcal{CR}(M, N)$ to denote the set of common refinements of models M and N. From this point on, when we refer to common refinement, we always mean observational refinement.

Refer to Figure 2.1 for the following example. MTS $\mathcal{F}$ specifies the writers policy for acquiring a read-write lock: i) readers exclude writers, ii) writers exclude readers, iii) at most one writer can have the lock at any given time, iv) the number of concurrent readers allowed is not known. We can merge $\mathcal{F}$ with the MTS $\mathcal{D}$ that states that there can be at least two concurrent readers. Model $\mathcal{H}$ is a common refinement of these models. Note that $\mathcal{D} \preceq_o \mathcal{H}\backslash\{$ $\texttt{getWriteLock}, \texttt{relWriteLock}\}$ holds via the relation

$$R = \{(\mathcal{D}_0, \mathcal{H}_0), (\mathcal{D}_1, \mathcal{H}_1), (\mathcal{D}_2, \mathcal{H}_2), (\mathcal{D}_0, \mathcal{H}_3)\},$$

and $\mathcal{F} \preceq_o \mathcal{H}$ holds via the relation

$$R = \{(\mathcal{F}_0, \mathcal{H}_0), (\mathcal{F}_1, \mathcal{H}_3), (\mathcal{F}_2, \mathcal{H}_2), (\mathcal{F}_2, \mathcal{H}_1)\}.$$

$\mathcal{H}$ is a refinement of $\mathcal{D}$ and $\mathcal{F}$ and thus can simulate the required behavior of both models. For example, like $\mathcal{D}$, $\mathcal{H}$ allows up to two readers to access the lock concurrently, e.g., it admits the

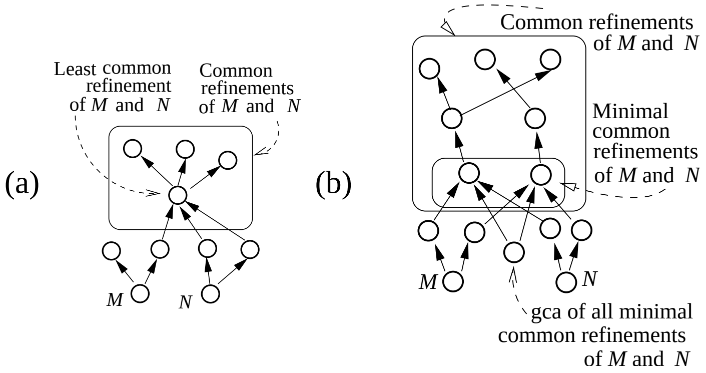
(a)
Least common refinement of M and N
Common refinements of M and N
M
N
(b)
Common refinements of M and N
Minimal common refinements of M and N
M
N
gca of all minimal common refinements of M and N

Figure 3.1: Common refinements for consistent models M and N: (a) M and N have the least common refinement; (b) M and N have no least common refinement.

trace

$$\texttt{getReadLock}, \texttt{getReadLock}, \texttt{relReadLock}, \texttt{relReadLock}, \ldots$$

As in model $\mathcal{F}$, $\mathcal{H}$ allows one writer to access the lock (e.g., it admits the trace getWriteLock, relWriteLock, ...). On the other hand, $\mathcal{D}$ and $\mathcal{F}$ can simulate the possible behaviour of $\mathcal{H}$, i.e., $\mathcal{H}$ cannot introduce behaviour that is proscribed in $\mathcal{D}$ or $\mathcal{F}$. If $\mathcal{H}$ had possible traces allowing concurrent access to the lock by readers and writers, (e.g., getWriteLock, getReadLock), then $\mathcal{H}$ would not be a refinement of $\mathcal{D}$ or $\mathcal{F}$, as such traces are not possible in either model.

3.2 Merge as a Minimal Common Refinement

Note that common refinement allows model $\mathcal{H}$ to proscribe traces that were possible in models $\mathcal{D}$ or $\mathcal{F}$. In other words, $\mathcal{H}$ may not be able to simulate the possible behaviour of $\mathcal{D}$ and $\mathcal{F}$. For example, the trace getReadLock, getReadLock, getReadLock, ..., which allows three or more concurrent readers, is a possible trace of $\mathcal{D}$ and $\mathcal{F}$, but is proscribed in $\mathcal{H}$. Consequently, the merged model $\mathcal{H}$ introduces knowledge that neither of the original models has, e.g., proscribing three or more concurrent readers. Instead, we prefer a least refined common refinement of the original models. For example, model $\mathcal{H}'$ is a common refinement of models $\mathcal{D}$ and $\mathcal{F}$ that, unlike $\mathcal{H}$, does not restrict the trace getReadLock, getReadLock, getReadLock.

Model $\mathcal{H}'$ is called the *least common refinement* of $\mathcal{D}$ and $\mathcal{F}$.

Definition 12. (Least Common Refinement) *An MTS P is the* least common refinement *(LCR) of modal transition systems M and N if $P \in \mathcal{CR}(M,N)$, $\alpha P = \alpha M \cup \alpha N$, and for any $Q \in \mathcal{CR}(M,N)$, $P \preceq_o Q@\alpha P$.*

Note that least common refinements are unique up to observational equivalence, and hence we refer to *the* least common refinement, denoted $\mathcal{LCR}_{M,N}$ for models M and N

An LCR of the original systems may not exist for two reasons. Firstly, it is possible that no common refinement exists. Secondly, a common refinement may exist, but there may be no least one. We discuss these possibilities below.

Consider the models in Figure 2.1(a). Models $\mathcal{A}$ and $\mathcal{B}$ do not have a common refinement: suppose some model P is a common refinement of $\mathcal{A}$ and $\mathcal{B}$, with $\alpha P = \alpha\mathcal{A} = \alpha\mathcal{B}$, and let $w = \texttt{getReadLock}, \texttt{getReadLock}$. We know that $\mathcal{B} \overset{w}{\Longrightarrow}_r \mathcal{B}_2$, and because $\mathcal{B} \preceq_o P$, there exists P' such that $P \overset{w}{\Longrightarrow}_r P'$. Since $\mathcal{A} \preceq_o P$, trace w should be possible in $\mathcal{A}$, which is a contradiction. In fact, model $\mathcal{A}$ is inconsistent with all models that give concurrent read access to more than one process, namely $\mathcal{B}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{H}$ and $\mathcal{H}'$. Merge can therefore only be defined for *consistent* models.

Definition 13. (Consistency) *MTSs M and N are* consistent *if there exists an MTS P such that P is a common refinement of M and N.*

Now refer to the models shown in Figure 3.2(a). Models $\mathcal{I}$ and $\mathcal{J}$ do have common refinements, e.g., $\mathcal{K}$, $\mathcal{L}$ and $\mathcal{O}$, but no LCR. Intuitively, to find $\mathcal{LCR}_{\mathcal{I},\mathcal{J}}$, we must refine $\mathcal{I}$ into $\mathcal{I}'$ so that $(\mathcal{I}' \setminus \{\texttt{a},\texttt{b}\}) \overset{\texttt{c}}{\Longrightarrow}_r$. Hence, we must transform the maybe transition on c in $\mathcal{I}$ to a required transition, and also transform one of the maybe transitions on a or b. If we transform all three transitions, we obtain the model $\mathcal{O}$. However, if we choose not to transform the transition either on a or on b, then we obtain the models $\mathcal{K}$ and $\mathcal{L}$, which are both refined by, but not equivalent to $\mathcal{O}$. These common refinements are not comparable (neither is a refinement

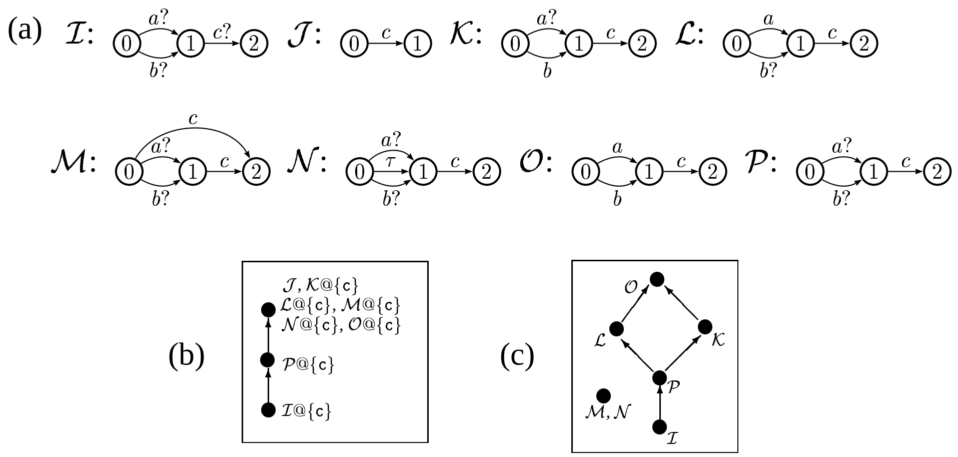

Figure 3.2: (a) Example MTSs; Relationships between models: (b) with respect to the alphabet $\{\mathtt{c}\}$; (c) with respect to the alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.

of the other) because of the different choices made on which non-shared maybe transition to make required.

It is not possible to find common refinements of $\mathcal{I}$ and $\mathcal{J}$ which are less refined than $\mathcal{K}$ and $\mathcal{L}$. For example, $\mathcal{P}$ is less refined than both but is not a refinement of $\mathcal{J}$. Hence, we refer to $\mathcal{K}$ and $\mathcal{L}$ as the *minimal common refinements* of $\mathcal{I}$ and $\mathcal{J}$. Note that models $\mathcal{M}$ and $\mathcal{N}$ are incorrect attempts of building minimal common refinements of $\mathcal{I}$ and $\mathcal{J}$. These are not refinements of $\mathcal{I}$ because they can both transit on $\mathtt{c}$ from the initial state through (a sequence of) required transitions, whereas $\mathcal{I}$ cannot do so from its initial state.

Definition 14. (Minimal Common Refinement) *An MTS P is a* minimal common refinement *(MCR) of MTSs M and N if $P \in \mathcal{CR}(M, N)$, $\alpha P = \alpha M \cup \alpha N$, and there is no MTS $Q \not\equiv_o P$ such that $Q \in \mathcal{CR}(M, N)$ and $Q@\alpha P \preceq_o P$.*

Remark 1. *$\mathcal{LCR}_{M,N}$ is also an MCR of M and N. In addition, if P is the only MCR of M and N (up to equivalence), then it is $\mathcal{LCR}_{M,N}$. Finally, if M and N are consistent, then they have an MCR.*

We write $\mathcal{MCR}(M, N)$ to denote the set of MCRs of models M and N. We are now ready to formally define merge.

Definition 15. (Merge) *The* merge *of two consistent MTSs M and N, written $M + N$, is either $\mathcal{LCR}_{M,N}$, it if exists, or* one *of the models in $\mathcal{MCR}(M, N)$.*

In Chapter 4, we give sufficient conditions for the uniqueness of merge (i.e., for the LCR to exist), and in Section 3.3, we discuss a model that can be used as a starting point for the case when merge is not unique.

3.3 Handling Multiple Merges

By definition of merge, if two models are consistent but have no LCR, their merge could result in *any* of their MCRs. However, any choice of MCR rules out the others. Hence, it is helpful to find a model that characterizes the point at which incompatible decisions must be made in order to merge the two models and produce an MCR. This model is the greatest common abstraction (GCA) of all MCRs: the most refined model from which we can arrive through refinement to any one of the MCRs.

Definition 16. (Greatest Common Abstraction) *Let M and N be consistent MTSs. An MTS Q is a* common abstraction *of all MCRs of M and N if $\alpha Q = \alpha M \cup \alpha N$ and for all $P \in \mathcal{MCR}(M, N)$, it holds that $Q \preceq_o P$. A common abstraction (CA) Q of all MCRs of M and N is the* greatest common abstraction *(GCA) if for any other common abstraction Q', it holds that $Q' \preceq_o Q$.*

Remark 2. *By Remark 1, $\mathcal{LCR}_{M,N}$ is also the GCA of all MCRs of M and N.*

We denote the GCA of all MCRS of M and N by $\mathcal{GCA}_{M,N}$.

Proposition 2. (Existence and Uniqueness of GCA) *The GCA between two MTSs always exists and is unique up to observational equivalence.*

Note that the GCA itself may not be a common refinement of the models being merged (see 3.1). For example, $\mathcal{GCA}_{\mathcal{I},\mathcal{J}}$ (see Figure 3.2(a)) is model $\mathcal{P}$. This model is not a refinement of $\mathcal{J}$, but could be refined to become one. Further, any refinement of this model rules out the possibility of arriving at one of the MCRs $\mathcal{K}$ or $\mathcal{L}$.

The relationship between models $\mathcal{I}$ through $\mathcal{N}$ of Figure 3.2(a) is shown in Figures 3.2(b) and 3.2(c). As in Figures 2.1(b) and 2.1(c), each graph relates models with the same alphabets. Since $\mathcal{J}$ does not have a or b in its alphabet, it cannot be compared to the other models unless these have their alphabets restricted to $\alpha\mathcal{J}$. Hence, $\mathcal{J}$ does not appear in Figure 3.2(c), where arrows depict observational refinement between models over the alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. However, $\mathcal{J}$ does appear in Figure 3.2(b), where the compared models have the alphabet $\{\mathtt{c}\}$.

3.4 Discussion

In conclusion, what should be the result of merging two consistent modal transition systems, M and N? If $\mathcal{LCR}_{M,N}$ exists, then this is the desired result of the merge. However, if M and N are consistent but their LCR does not exist, then the merge process should result in one of the MCRs of M and N. Model merging should support the modeller in choosing which MCR is the most appropriate. This can be done by producing $\mathcal{GCA}_{M,N}$ and supporting its elaboration to produce a particular MCR (see [49]). This would allow the modeller to choose, possibly after validating with stakeholders, which is the appropriate way of combining two different descriptions of the behaviour of the same system.

Chapter 4

Characterizing Merge

In this chapter, we address several questions related to merge: how can the consistent behaviours from two consistent models be described? When is the merge of consistent models unique? What temporal properties are preserved in merge? How can we use temporal properties to determine if two systems can be merged?

Section 4.1 defines consistency relations between two models that captures the notion of a common refinement, and discusses the role of the largest such relation. Section 4.2 gives sufficient conditions for the uniqueness of merge. Section 4.3 proves that refinement is logically characterized by $\mathcal{L}_\mu^w$ and preserves FLTL, and discusses the practical implications of these results for modellers. Finally, Section 4.4 uses $\mathcal{L}_\mu^w$ properties to characterize consistency.

4.1 Consistency Relations

In this section, we define and discuss issues related to consistency relations. These relations are used throughout the thesis: in Section 4.2 to describe when merge is unique, in Section 4.4 to characterize consistency in terms of $\mathcal{L}_\mu^w$ properties, and in Chapter 6 to define two merge algorithms.

In order to merge two consistent models, it is necessary to understand precisely which of their behaviours can be integrated. In particular, a state in any common refinement of two

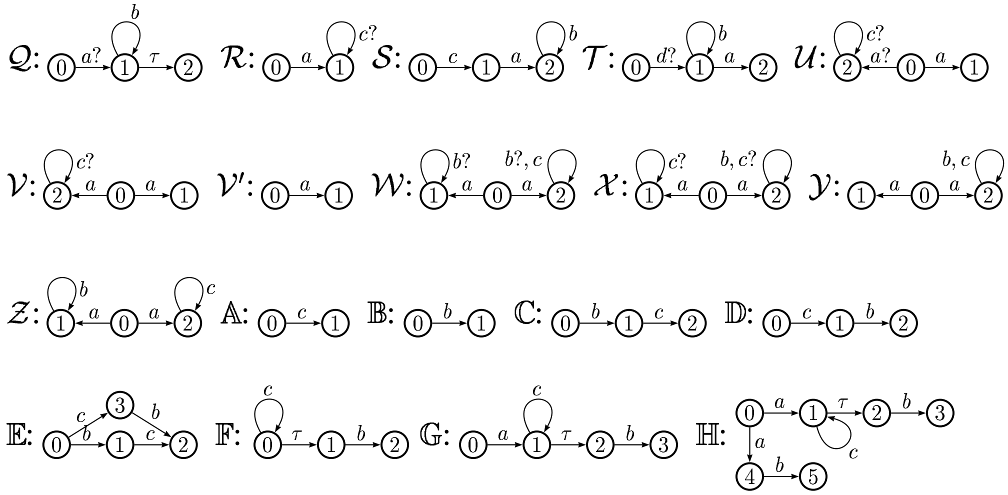

Figure 4.1: Example MTSs.

models is intuitively a combination of two consistent states: one from each of the original models. In $M = (S_M,\ L_M,\ \Delta^r_M,\ \Delta^p_M,\ s_{0M})$ and $N = (S_N,\ L_N,\ \Delta^r_N,\ \Delta^p_N,\ s_{0N})$, states $s \in S_M$ and $t \in S_N$ are consistent if and only if there is a common refinement of M_s and N_t. Therefore, $N_t @ \alpha M$ should be able to simulate required behaviour at M_s with possible behaviour, and vice-versa. A *consistency relation*, which is similar to a two-way refinement relation, is used to describe such pairs of reachable consistent states of two models.

Definition 17. (Consistency Relation) *A* consistency relation *between MTSs M and N is a binary relation $C_{MN} \subseteq \wp \times \wp$ such that $(M, N) \in C_{MN}$, and the following conditions hold for all $\ell \in Act_\tau$ and all $(M, N) \in C_{MN}$:*

1. $M \overset{\hat{\ell}}{\Longrightarrow}_r M' \wedge \ell \in \alpha N \cup \{\tau\} \Rightarrow \exists x_1, x_2, \ldots, x_i, y_1, y_2, \ldots, y_j \in (\mathrm{Act}_\tau \setminus \alpha M) \cdot$
   $N \overset{\hat{x_1}}{\Longrightarrow}_p N_{x_1} \cdots \overset{\hat{x_i}}{\Longrightarrow}_p N_{x_i} \overset{\hat{\ell}}{\Longrightarrow}_p N_{y_1} \overset{\hat{y_1}}{\Longrightarrow}_p \cdots \overset{\hat{y_j}}{\Longrightarrow}_p N' \wedge$
   $(M', N') \in C_{MN} \wedge \forall k \cdot (M, N_{x_k}) \in C_{MN} \wedge \forall k \cdot (M', N_{y_k}) \in C_{MN}$

2. $N \overset{\hat{\ell}}{\Longrightarrow}_r N' \wedge \ell \in \alpha M \cup \{\tau\} \Rightarrow \exists x_1, x_2, \ldots, x_i, y_1, y_2, \ldots, y_j \in (\mathrm{Act}_\tau \setminus \alpha N) \cdot$
   $M \overset{\hat{x_1}}{\Longrightarrow}_p M_{x_1} \cdots \overset{\hat{x_i}}{\Longrightarrow}_p M_{x_i} \overset{\hat{\ell}}{\Longrightarrow}_p M_{y_1} \overset{\hat{y_1}}{\Longrightarrow}_p \cdots \overset{\hat{y_j}}{\Longrightarrow}_p M' \wedge$
   $(M', N') \in C_{MN} \wedge \forall k \cdot (M_{x_k}, N) \in C_{MN} \wedge \forall k \cdot (M_{y_k}, N') \in C_{MN}$

3. $M \overset{\ell}{\Longrightarrow}_r M' \wedge \ell \notin \alpha N \cup \{\tau\} \Rightarrow \exists N' \cdot N \overset{\epsilon}{\Longrightarrow}_p N' \wedge (M', N') \in C_{MN}$

4. $N \overset{\ell}{\Longrightarrow}_r N' \wedge \ell \notin \alpha M \cup \{\tau\} \Rightarrow \exists M' \cdot M \overset{\epsilon}{\Longrightarrow}_p M' \wedge (M', N') \in C_{MN}$

Hence, there is a consistency relation between M and N if whenever M can transit on a shared action ℓ through a required transition to M' ($M \overset{\hat{\ell}}{\Longrightarrow}_r M'$), then $N@\alpha M$ can transit on a possible transition to N' ($N@\alpha M \overset{\hat{\ell}}{\Longrightarrow}_p$) such that M' and N' are consistent. However, this means that N can transit on ℓ, possibly preceded and succeeded by zero or more observable transitions on actions not in $\alpha M \cup \{\tau\}$. In order for $M \overset{\hat{\ell}}{\Longrightarrow}_r M'$ and $N@\alpha M \overset{\hat{\ell}}{\Longrightarrow}_p N'$ to be consistent, the successors of N preceding ℓ must be consistent with M, and the successors of N succeeding ℓ must be consistent with M' (Condition (1)). The case when N can transit on a shared action through a required transition is analogous to Condition (1). If M can transit on a non-shared action ℓ through a required transition, then N can internally transit through possible transitions to some N' that is consistent with M' (Condition (3)), since, as discussed, transitions out of the scope of one system are in essence the same as internal activity to that system. Finally, the case when N transits on a required transition is analogous (Condition (4)).

For example, consider the models in Figure 4.1 and assume that $\alpha\mathcal{Q} = \{\mathtt{a}, \mathtt{b}\}$ and $\alpha\mathcal{R} = \alpha\mathcal{S} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. $C_{\mathcal{QR}} = \{(\mathcal{Q}_0, \mathcal{R}_0), (\mathcal{Q}_2, \mathcal{R}_1)\}$ is a consistency relation between $\mathcal{Q}$ and $\mathcal{R}$. The transition $\mathcal{R}_0 \xrightarrow{a}_r \mathcal{R}_1$ is matched in $\mathcal{Q}$ with $\mathcal{Q}_0 \overset{a}{\Longrightarrow}_m \mathcal{Q}_2$, and *cannot* be matched with $\mathcal{Q}_0 \xrightarrow{a}_m \mathcal{Q}_1$ because $\mathcal{Q}_1$ and $\mathcal{R}_0$ are inconsistent on $\mathtt{b}$, violating Condition (1) in Definition 17. This matching makes sense because no common refinement of $\mathcal{Q}$ and $\mathcal{R}$ includes a transition on $\mathtt{b}$ (e.g., $\mathcal{R} \in \mathcal{CR}(\mathcal{Q}, \mathcal{R})$). On the other hand, there is no consistency relation between models $\mathcal{Q}$ and $\mathcal{S}$. Suppose $C_{\mathcal{QS}}$ was such a relation. Since $\mathcal{S}_0 \xrightarrow{c}_r \mathcal{S}_1$ and $\mathtt{c} \notin \alpha\mathcal{Q} \cup \{\tau\}$, the pair $(\mathcal{Q}_0, \mathcal{S}_1)$ must be in $C_{\mathcal{QS}}$ (Condition (4)). $\mathcal{S}_1 \xrightarrow{a}_r \mathcal{S}_2$ must be identified with $\mathcal{Q}_0 \xrightarrow{a}_m \mathcal{Q}_1$ because $\mathcal{S}_2$ requires $\mathtt{b}$ and $\mathcal{Q}_2$ proscribes it, and hence $(\mathcal{Q}_1, \mathcal{S}_2)$ is in $C_{\mathcal{QS}}$ (Condition (2)). The only internal behaviour at $\mathcal{S}_2$ is a self-loop on ϵ, which forces $\mathcal{Q}_1 \xrightarrow{\tau}_r \mathcal{Q}_2$ to be matched with $\mathcal{S}_2 \overset{\epsilon}{\Longrightarrow}_p \mathcal{S}_2$ (Condition (1)), i.e., $(\mathcal{Q}_2, \mathcal{S}_2)$ must be in $C_{\mathcal{QS}}$. However, since $\mathcal{S}_2$ requires $\mathtt{b}$ and $\mathcal{Q}_2$ proscribes all actions, Condition (2) is violated, and therefore there is no consistency relation between $\mathcal{Q}$ and $\mathcal{S}$. Note that there is also no common refinement of these models.

Conditions (1) and (2) in Definition 17 require that the intermediate states are identified in the consistency relation, e.g., $(M, N_{x_i}) \in C_{MN}$ and $(M', N_{y_i}) \in C_{MN}$ in Condition (1). For example, refer to models $\mathcal{R}$ and $\mathcal{T}$ in Figure 4.1 with $\alpha\mathcal{R} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $\alpha T = \{\mathtt{a}, \mathtt{b}, \mathtt{d}\}$. There is no consistency relation between these models because the only possible match for the transition $\mathcal{R}_0 \xrightarrow{a}_r \mathcal{R}_1$ is the sequence $\mathcal{T}_0 \xrightarrow{d}_m \mathcal{T}_1 \xrightarrow{a}_r \mathcal{T}_2$, where $\mathtt{d} \notin \alpha\mathcal{R}$. However, $\mathcal{R}_1$ and $\mathcal{T}_1$ are inconsistent on $\mathtt{b}$, violating Condition (1), and resulting in inconsistencies between these models. In particular, Conditions (1) and (2) guarantee that when a single observable transition in one model is identified with a sequence of observable transitions in the other model, possibly on non-shared actions, then the states corresponding to the transitions on non-shared actions are not sources of inconsistencies.

We now prove that our notion of the consistency relation is complete, i.e., consistent models are precisely those that have a consistency relation between them.

Theorem 2. (Consistency Relations Characterize Consistency) *Two MTSs are consistent if and only if there is a consistency relation between them.*

Proof:

We illustrate the case for equal vocabularies; the case for different vocabularies is similar. Let M and N be MTSs such that $\alpha M = \alpha N$.

($\Leftarrow$) Assume M and N are consistent and let $Q \in \mathcal{CR}(M, N)$. $M \preceq_o Q$ with refinement relation R_M, and $N \preceq_o Q$ with refinement relation R_N. Define C_{MN} as follows:

$$C_{MN} = \{(M, N) \mid \exists Q_i \cdot (M, Q_i) \in R_M \wedge (N, Q_i) \in R_N\}. \tag{4.1}$$

We show that C_{MN} is a consistency relation between M and N by showing that Conditions (1) and (2) in Definition 17 hold. Clearly $(M, N) \in C_{MN}$. Let $\ell \in (\alpha M \cap \alpha N) \cup \{\tau\}$.

$$M \overset{\hat{\ell}}{\Longrightarrow}_r M' \Rightarrow \exists N' \cdot N \overset{\hat{\ell}}{\Longrightarrow}_p N' \wedge (M', N') \in C_{MN}$$

(Equation (4.1))

$$\begin{aligned}
\Leftarrow\quad & M \overset{\hat{\ell}}{\Longrightarrow}_r M' \Rightarrow \exists N' \cdot N \overset{\hat{\ell}}{\Longrightarrow}_p N' \wedge \exists Q' \cdot (M', Q') \in R_M \wedge (N', Q') \in R_N \\
& (N \preceq_o Q) \\
\Leftarrow\quad & M \overset{\hat{\ell}}{\Longrightarrow}_r M' \Rightarrow \exists Q' \cdot Q \overset{\hat{\ell}}{\Longrightarrow}_r Q' \wedge (M', Q') \in R_M \\
& (M \preceq_o Q) \\
=\quad & \mathbf{t}.
\end{aligned}$$

Condition (2) is proven in a similar fashion.

($\Rightarrow$) This part of the proof is constructive, utilizing the $+_{cr}$ operator presented in Section 6.4 for building a common refinement. In particular, if there is a consistency relation between M and N, then by Theorem 11, $M +_{cr} N \in \mathcal{CR}(M, N)$. □

There is a notion of the *largest* consistency relation between two consistent models, which is defined as the union of all consistency relations between them. Intuitively, the largest consistency relation describes all reachable consistent behaviours between two consistent models. For example, consider $\mathcal{R}$ and $\mathcal{U}$ in Figure 4.1 over the vocabulary $\{\mathsf{a}, \mathsf{c}\}$. $C_{\mathcal{RU}} = \{(\mathcal{R}_0, \mathcal{U}_0), (\mathcal{R}_1, \mathcal{U}_1), (\mathcal{R}_1, \mathcal{U}_2))\}$ is the largest consistency relation between them, but $C'_{\mathcal{RU}} = C_{\mathcal{RU}} \setminus \{(\mathcal{R}_1, \mathcal{U}_2)\}$ is also a consistency relation. These two relations give rise to different common refinements of $\mathcal{R}$ and $\mathcal{U}$, namely $\mathcal{V}$ and $\mathcal{V}'$. Unlike $\mathcal{V}'$, model $\mathcal{V}$ does not rule out the possibility of action c occurring after action a. We discuss this issue in more depth in Section 6.2, where we give an algorithm for building the largest consistency relation between two models.

4.2 Uniqueness

We now give sufficient conditions for the merge of two consistent systems to be unique.

Recall that models $\mathcal{I}$ and $\mathcal{J}$ in Figure 3.2 do not have an LCR due to the fact that in any LCR, at least one non-shared maybe transition in $\mathcal{I}$ must be transformed into a required transition, yet multiple incompatible choices of which transition to transform are available.

The following condition restricts the existence of such choices.

Definition 18. (Multiple-Maybe Condition) *Let M and N be MTSs and let C_{MN} be the largest consistency relation between them. M and N satisfy the* multiple-maybe *condition if it is* **not** *the case that for all $(M, N) \in C_{MN}$, Conditions (1) – (3) or Conditions (4) – (6) simultaneously hold, where $\ell \in \alpha M \cap \alpha N$:*

1. $N \xRightarrow{\ell}_r \wedge M \overset{\ell}{\not\Longrightarrow}$,
2. $\forall M' \cdot \exists w \in (\alpha M \setminus \alpha N)^* \cdot M \xRightarrow{w}_\gamma M' \xRightarrow{\ell}_p \wedge (M', N) \in C_{MN} \Rightarrow (\gamma = m)$,
3. $\exists M', M'' \cdot \exists w_1, w_2 \in (\alpha M \setminus \alpha N)^* \cdot (w_1 \neq w_2) \wedge (M \xRightarrow{w_1}_m M' \xRightarrow{\ell}_p)$
$\wedge (M \xRightarrow{w_2}_m M'' \xRightarrow{\ell}_p) \wedge (M', N) \in C_{MN} \wedge (M'', N) \in C_{MN}$.

Conditions (4) – (6) are analogous to (1) – (3) with the roles of M and N reversed.

In Condition (1), M proscribes the shared action ℓ, whereas N requires it. Condition (2) says that if M' is reachable from M through one or more non-shared actions, and M' can transition on ℓ, then M' is only reachable via non-shared *maybe* behaviour. Condition (3) guarantees that there are at least two different $w's$ and M''s in Condition (2), i.e., two different non-shared maybe paths to states in M that can transition on ℓ. If these three conditions hold, then for any P in $\mathcal{MCR}(M, N)$, some non-shared maybe behaviours in M must be required behaviours in P in order to guarantee that $P@\alpha N \xRightarrow{c}_r$ (i.e, that P refines N). However, multiple choices of which maybe behaviours to convert to required behaviours in P exist, and therefore multiple non-equivalent MCRs exist. Finally, Conditions (4) – (6) are the same as Conditions (1) – (3), except the choices are made in N instead of M.

Consider models $\mathcal{W}$ and $\mathcal{X}$ in Figure 4.1 over the vocabulary $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. Both $\mathcal{W}$ and $\mathcal{X}$ have two non-equivalent successors on a from the initial state, i.e., $\mathcal{W} \xrightarrow{a}_r \mathcal{W}_1$ and $\mathcal{W} \xrightarrow{a}_r \mathcal{W}_2$ such that $\mathcal{W}_1 \not\equiv_o \mathcal{W}_2$, and similarly for $\mathcal{X}$. However, both $\mathcal{W}_1$ and $\mathcal{W}_2$ are consistent with $\mathcal{X}_1$ and $\mathcal{X}_2$. In particular, $\mathcal{Y}$ is an MCR of $\mathcal{W}$ and $\mathcal{X}$ that corresponds to combining $\mathcal{W}_1$ with $\mathcal{X}_1$, and $\mathcal{W}_2$ with $\mathcal{X}_2$, whereas $\mathcal{Z}$ corresponds to combining $\mathcal{W}_1$ with $\mathcal{X}_2$, and $\mathcal{W}_2$ with $\mathcal{X}_1$. $\mathcal{Y} \not\equiv_o \mathcal{Z}$, and therefore $\mathcal{LCR}_{\mathcal{W},\mathcal{X}}$ does not exist. This example illustrates that different MCRs may arise when there are multiple ways of combining non-deterministic behaviour on a shared action.

Definition 19. (Determinacy Condition) *Let M and N be MTSs, C_{MN} be the largest consistency relation between them, and $X = \alpha M \cap \alpha N$. M and N satisfy the* determinacy condition *if for all $(M, N) \in C_{MN}$, it is* **not** *the case that for some $\ell \in X \cup \{\tau\}$:*

1. *$M@X \overset{\hat{\ell}}{\Longrightarrow}_p M' \wedge M@X \overset{\hat{\ell}}{\Longrightarrow}_p M''@X \wedge M' \not\equiv_o M''$,*

2. *$N@X \overset{\hat{\ell}}{\Longrightarrow}_p N'@X \wedge N@X \overset{\hat{\ell}}{\Longrightarrow}_p N''@X \wedge N' \not\equiv_o N''$,*

3. *$\{(M', N'), (M'', N'')\} \subseteq C_{MN} \wedge ((M', N'') \in C_{MN} \vee (M'', N') \in C_{MN})$.*

Intuitively, if the determinacy condition is violated at (M, N), then without loss of generality, there is a successor M' of M via a shared action ℓ that is consistent with at least two non-equivalent successors of N via ℓ, say N' and N''. In addition, one of these successors, say N'', is consistent with a successor M'' of M via ℓ that is not equivalent to M'. Hence, if M' is combined with N', then M' may or may not be combined with N'', because N'' could be combined with M'' instead. Since $M' \not\equiv_o M''$ and $N' \not\equiv_o N''$, the different combinations of non-determinism may lead to non-equivalent behaviours in $M' + N'$ and $M' + N''$, and hence to the existence of multiple incomparable MCRs.

Finally, consider models $\mathbb{A}$ and $\mathbb{B}$ in Figure 4.1 and assume that $\alpha\mathbb{A} = \{\mathtt{c}\}$ and $\alpha\mathbb{B} = \{\mathtt{b}\}$. Models $\mathbb{C}$, $\mathbb{D}$, and $\mathbb{E}$ over the vocabulary $\{\mathtt{b}, \mathtt{c}\}$ are in $\mathcal{MCR}(\mathbb{A}, \mathbb{B})$, yet none of them are equivalent. Intuitively, $\mathcal{LCR}_{\mathbb{A},\mathbb{B}}$ does not exist because both models can transit on non-shared actions: $\mathbb{A} \xrightarrow{c}_r$ and $\mathbb{B} \xrightarrow{b}_r$, where $\mathtt{c} \in \alpha\mathbb{A} \setminus \alpha\mathbb{B}$ and $\mathtt{b} \in \alpha\mathbb{B} \setminus \alpha\mathbb{A}$. Hence, choices exist on the order in which these actions appear in an MCR: the MCR $\mathbb{C}$ corresponds to putting the transition on $\mathtt{b}$ first, the MCR $\mathbb{D}$ corresponds to putting the transition on $\mathtt{c}$ first, and the MCR $\mathbb{E}$ corresponds to allowing both orders. The following condition restricts the existence of such choices.

Definition 20. (Non-Shared Condition) *Let M and N be MTSs and let C_{MN} be the largest consistency relation between them. M and N satisfy the* non-shared condition *if for all $(M, N) \in C_{MN}$, it is* **not** *the case that $M \xRightarrow{\ell_1}_p$ and $N \xRightarrow{\ell_2}_p$, where $\ell_1 \in \alpha M \setminus \alpha N$ and $\ell_2 \in \alpha N \setminus \alpha M$.*

The three conditions defined in this section are sufficient to prove the existence of the LCR between two consistent models.

Theorem 3. (Sufficient conditions for the existence of LCR) *If M and N are consistent MTSs that satisfy the multiple-maybe condition, the determinacy condition, and the non-shared condition, then $\mathcal{LCR}_{M,N}$ exists.*

Proof:

Let M and N be consistent MTSs with $\alpha M = \alpha N$, and let C_{MN} be the largest consistency relation between them. Assume there exist P and Q in $\mathcal{MCR}(M, N)$ such that $P \not\equiv_o Q$. Since $P \not\preceq_o Q$, there are two possible cases:

1. $\exists \ell \in \alpha M \cap \alpha N \cup \{\tau\} \cdot \exists P' \cdot P \xrightarrow{\ell}_r P' \wedge \forall Q' \cdot (Q \xRightarrow{\hat{\ell}}_r Q' \Rightarrow P' \not\preceq_o Q')$:

$$P \xrightarrow{\ell}_r P' \wedge P \in \mathcal{MCR}(M, N)$$

(Definitions 5 and 14)

$$\Rightarrow \quad \exists M', N' \cdot M \xRightarrow{\hat{\ell}}_p M' \wedge N \xRightarrow{\hat{\ell}}_p N' \wedge M' \preceq_o P' \wedge N' \preceq_o P' \wedge P' = M' + N'$$

$(P' \in \mathcal{CR}(M', N'))$

$$\Rightarrow \quad \exists M', N' \cdot M \xRightarrow{\hat{\ell}}_p M' \wedge N \xRightarrow{\hat{\ell}}_p N' \wedge (M', N') \in C_{MN} \wedge P' = M' + N'.$$

Without loss of generality, assume $M \xRightarrow{\hat{\ell}}_r M'$, otherwise P would not require ℓ.

$$M \xRightarrow{\hat{\ell}}_r M' \wedge Q \in \mathcal{MCR}(M, N)$$

(Definitions 5 and 14)

$$\Rightarrow \quad \exists Q' \cdot Q \xRightarrow{\hat{\ell}}_r Q' \wedge M' \preceq_o Q' \wedge Q \in \mathcal{MCR}(M, N)$$

$$
\begin{aligned}
& \text{(Definitions 5 and 14)} \\
\Rightarrow\quad & \exists Q', N' \cdot Q \overset{\hat{\ell}}{\Longrightarrow}_r Q' \wedge M' \preceq_o Q' \wedge N \overset{\hat{\ell}}{\Longrightarrow}_p N'' \wedge N'' \preceq_o Q' \wedge Q' = M' + N'' \\
& (Q' \in \mathcal{CR}(M', N'')) \\
\Rightarrow\quad & \exists Q', N' \cdot Q \overset{\hat{\ell}}{\Longrightarrow}_r Q' \wedge N \overset{\hat{\ell}}{\Longrightarrow}_p N'' \wedge (M', N'') \in C_{MN} \wedge Q' = M' + N''.
\end{aligned}
$$

We can assume $\mathcal{LCR}_{M',N'}$ and $\mathcal{LCR}_{M'',N'}$ both exist; otherwise, we repeat the proof for $M = M'$ and $N = N'$ or $M = M'$ and $N = N''$ until they do. Hence:

$$
\begin{aligned}
& M' + N' = P' \wedge Q' = M' + N'' \\
& \text{(By (1))} \\
\Rightarrow\quad & M' + N' = P' \not\preceq_o Q' = M' + N'' \\
& \text{(Proposition 4 in Section 5.1, since } P' \text{ and } Q' \text{ are LCRs)} \\
\Rightarrow\quad & N' \not\equiv_o N''.
\end{aligned}
$$

In addition, there is no Q'' such that $Q \overset{\hat{\ell}}{\Longrightarrow}_r Q''$ and $Q'' = M' + N'$, otherwise $P' \equiv_o Q''$, contradicting (1). Since $(M', N') \in C_{MN}$, there exists M'' such that $M \overset{\hat{\ell}}{\Longrightarrow}_p M''$ and $(M'', N') \in C_{MN}$ (i.e., in Q, N' is combined with M'' instead of M'). Hence, $M' \not\equiv_o M''$; otherwise, $P' = M' + N' \equiv_o M'' + N' = Q'$, contradicting (1). Putting the above together:

$$
\begin{aligned}
& M \overset{\hat{\ell}}{\Longrightarrow}_r M' \wedge M \overset{\hat{\ell}}{\Longrightarrow}_p M'' \wedge M' \not\equiv_o M'' \wedge N \overset{\hat{\ell}}{\Longrightarrow}_p N' \wedge N \overset{\hat{\ell}}{\Longrightarrow}_p N'' \wedge N' \not\equiv_o N'' \\
& \wedge \{(M', N'), (M', N''), (M'', N')\} \subseteq C_{MN},
\end{aligned}
$$

which violates the determinacy condition.

2. $\exists \ell \in \alpha M \cap \alpha N \cup \{\tau\} \cdot \exists Q' \cdot Q \xrightarrow{\ell}_p Q' \wedge \forall P' \cdot (P \overset{\hat{\ell}}{\Longrightarrow}_p P' \Rightarrow P' \not\preceq_o Q')$:

   Analogous to Case 1.

When $\alpha M \neq \alpha N$, the proof is similar, but with additional subcases in Cases 1 and 2 above, depending on whether ℓ is a shared or non-shared action. In Case 1 above, if ℓ is a shared action, either the determinacy condition or the multiple-maybe condition is violated, and if ℓ is a non-shared action, the non-shared condition is violated. Case 2 is analogous. □

Note that the conditions in Theorem 3 are reasonable because they are consequences of the modelling formalism. Specifically, the multiple-maybe condition is simply a consequence of allowing multiple non-equivalent paths to the same state. The non-shared condition is a consequence of the interleaving nature of MTSs, i.e., it is impossible to express simultaneous transitions in an MTS, such as "M transits on a and b", without specifying the order in which the actions occur. Finally, non-determinism in MTSs is intuitively due to different choices of how to abstract part of the system away. As these choices cannot be guaranteed to be equivalent, non-determinism in both models on the same action may lead to multiple MCRs, as in the determinacy condition.

It should also be noted that the multiple-maybe condition and the non-shared condition are necessary conditions: if either one is violated, then the LCR does not exist. On the other hand, the determinacy condition is sufficient but not necessary. This is due to the fact that $N' \not\equiv_o N''$ does not imply that $M' + N' \not\equiv_o M' + N''$. Therefore, different choices of how to combine non-deterministic behaviour may lead to equivalent behaviours, even if the successors are not equivalent. We leave for future work strengthening the determinacy condition to be a necessary condition as well.

4.3 Property Preservation

The logic $\mathcal{L}_\mu^w$ characterizes observational refinement and therefore merge. In the 3-valued world, this means that an MTS M is refined by an MTS N if and only if all *true* and *false* $\mathcal{L}_\mu^w$ properties in M are preserved in N.

Theorem 4. ($\mathcal{L}_\mu^w$ Characterizes Refinement) *If M and N are MTSs over the same vocabulary with initial states s_{0M} and s_{0N}, then:*

$$M \preceq_o N \Leftrightarrow \forall \phi \in \mathcal{L}_\mu^w \cdot (s_{0M} \in [\![\phi]\!]^{\mathbf{t}} \Rightarrow s_{0N} \in [\![\phi]\!]^{\mathbf{t}}) \wedge (s_{0M} \in [\![\phi]\!]^{\mathbf{f}} \Rightarrow s_{0N} \in [\![\phi]\!]^{\mathbf{f}})$$

Proof:

Follows from the fact that $\mathcal{L}_\mu$ characterizes strong refinement [29], $M \preceq_o N$ if and only if $M_\epsilon \preceq_o N_\epsilon$, and $\phi \in \mathcal{L}_\mu^w$ is *true* (*false*) in an MTS M if and only if $\phi' \in \mathcal{L}_\mu$ is *true* (*false*) in M_ϵ, where ϕ' is obtained from ϕ by replacing every occurrence of a weak next operator with its strong counterpart. □

Remark 3. *Theorem 4 does not give insight into preservation of maybe properties: they can become true, false, or remain maybe in the more refined model.*

To illustrate Theorem 4, refer to the models in Figure 4.1. $\mathcal{W} \preceq_o \mathcal{Y}$ with refinement relation $R = \{(\mathcal{W}_0, \mathcal{Y}_0), (\mathcal{W}_1, \mathcal{Y}_1), (\mathcal{W}_2, \mathcal{Y}_2)\}$. It can be verified that all *true* and *false* properties in $\mathcal{W}$ are preserved in $\mathcal{Y}$ and vice-versa (e.g., $\langle a \rangle_o \langle c \rangle_o \mathbf{t}$ and $[a]_o \langle a \rangle_o \mathbf{f}$). On the other hand, $\mathcal{Y} \not\preceq_o \mathcal{W}$ since $\mathcal{W}$ can still perform a transition on b after every transition on a, whereas $\mathcal{Y}_0 \xrightarrow{a}_r \mathcal{Y}_1$ and $\mathcal{Y}_1 \overset{b}{\not\Longrightarrow}$. In particular, $\langle a \rangle_o \langle b \rangle_o \mathbf{f}$ is *true* in $\mathcal{Y}$ and only *maybe* in $\mathcal{W}$.

The logic FLTL is also preserved under refinement. We begin with the following lemma.

Lemma 1. (Preservation of Trace Values) *If M and N are MTS's over the same vocabulary:*

$$M \preceq_o N \Rightarrow \forall \pi \in \Pi \cdot M(\pi) \leq_{inf} N(\pi)$$

The above lemma states that the information ordering on the values of traces is preserved under refinement, which is fundamental for proving preservation of any linear time logic.

Theorem 5. (Preservation of FLTL) *If M and N are MTSs over the same vocabulary:*

$$M \preceq_o N \Rightarrow \forall \phi \in \text{FLTL} \cdot \|\phi\|^M \leq_{inf} \|\phi\|^N.$$

Proof:

Suppose $M \preceq_o N$ and let $\phi \in$ FLTL.

$$(\|\phi\|^M = \mathbf{t}) \wedge (\|\phi\|^N \leq \bot)$$

(Definition 10)

$$= \quad (\|\phi\|^M = \mathbf{t}) \wedge (\exists \pi \in \Pi \cdot N(\pi) \geq \bot \wedge \pi \not\models \phi)$$

(Definition 10 and $N(\pi) \geq \bot \Rightarrow M(\pi) \geq \bot$ by Lemma 1)

$\Rightarrow \quad (\forall \pi \in \Pi \cdot M(\pi) \geq \bot \Rightarrow \pi \models \phi) \wedge (\exists \pi \in \Pi \cdot M(\pi) \geq \bot \wedge \pi \not\models \phi)$

(Law of Contradiction)

$= \quad \mathbf{f}.$

$\|\phi\|^M = \mathbf{f}$

(Definition 10)

$= \quad \exists \pi \in \Pi \cdot M(\pi) = \mathbf{t} \wedge \pi \not\models \phi$

($M(\pi) = \mathbf{t} \Rightarrow N(\pi) = \mathbf{t}$ by Lemma 1)

$\Rightarrow \quad \exists \pi \in \Pi \cdot N(\pi) = \mathbf{t} \wedge \pi \not\models \phi$

(Definition 10)

$= \quad \|\phi\|^N = \mathbf{f}.$

□

Although FLTL is preserved by refinement, it does not characterize it (adapted from [19]). For example, refer to models $\mathbb{B}$, $\mathbb{F}$, $\mathbb{G}$, and $\mathbb{H}$ in Figure 4.1 all over the vocabulary $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Suppose that FLTL characterizes refinement, i.e., if $M \not\preceq_o N$, there is an FLTL property that is *true* in M and *false* in N. Since $\mathbb{B} \not\preceq_o \mathbb{F}$, there exists ϕ in FLTL that is *true* in $\mathbb{B}$ and *false* in $\mathbb{F}$. By the fact that $\mathbb{H}_4 = \mathbb{B}$ and $\mathbb{G}_1 = \mathbb{F}$, the property $\mathbf{F}\phi$ is *true* in $\mathbb{G}$ and *false* in $\mathbb{H}$. However, $\mathbb{G} \equiv_o \mathbb{H}$, contradicting Theorem 5. Hence, FLTL does not characterize refinement.

Theorems 4 and 5 are fundamental to understanding the properties of merge from a stakeholder's point of view. In particular, properties that are invariant under stuttering [1] (i.e., eventualities, invariants, and so forth) are guaranteed to be preserved, regardless of the underlying semantics of the logic in which they are expressed. However, properties that rely on precise amounts of internal activity, i.e., immediate nexts or counting properties, cannot be guaranteed to be preserved by a merge. The case study in Chapter 7 illustrates the utility of these results on a realistic example.

4.4 Characterizing Consistency with Temporal Properties

In this section, we use Theorem 4 to characterize the case when two models can be merged using properties that *distinguish* between two systems, i.e., that evaluate to *true* in one system and *false* in the other.

As two models are consistent if and only if they have a common refinement, models over the same vocabulary must agree on all concrete behaviours, i.e., there should be no distinguishing $\mathcal{L}_\mu^w$ property between them.

Theorem 6. (Consistency Characterization: Same Vocabularies) *Two MTSs over the same vocabulary are consistent if and only if no $\mathcal{L}_\mu^w$ property distinguishes them.*

Proof:

($\Rightarrow$) If M and N are consistent with MCR P, and there is $\phi \in \mathcal{L}_\mu^w$ that distinguishes them, then ϕ would be *true* and *false* in P by Theorem 4.

($\Leftarrow$) If M and N are inconsistent, then by Theorem 10 of Section 6.3, Algorithm 2 in Figure 6.4 returns $\phi \in \mathcal{L}_\mu^w$ that distinguishes them. □

For example, $\mathcal{Y}$ and $\mathcal{Z}$ in Figure 4.1 are inconsistent because after every transition on $\mathtt{a}$ in $\mathcal{Z}$, a transition on either $\mathtt{b}$ or $\mathtt{c}$ is possible, whereas $\mathcal{Y}_0 \xrightarrow{a}_r \mathcal{Y}_1$ and $\mathcal{Y}_1$ proscribes both $\mathtt{b}$ and $\mathtt{c}$. The property $[a]_o(\langle b\rangle_o\mathbf{t} \vee \langle c\rangle_o\mathbf{t})$ is *false* in $\mathcal{Y}$ and *true* in $\mathcal{Z}$.

Before a distinguishing property is meaningful for models over different vocabularies, both models must first be restricted to their shared vocabulary, because the presence of non-shared actions may cause a property to distinguish between consistent systems. For example, consider models $\mathbb{B}$ and $\mathbb{D}$ in Figure 4.1 and assume that $\alpha\mathbb{B} = \{\mathtt{b}\}$ and $\alpha\mathbb{D} = \{\mathtt{b}, \mathtt{c}\}$. These models are consistent because $\mathbb{D}$ is a common refinement, but the property $\langle b\rangle_o\mathbf{t}$ is *true* in $\mathbb{B}$ and *false*

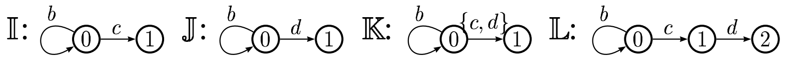

Figure 4.2: Inconsistent MTSs $\mathbb{I}$ and $\mathbb{J}$; $\mathbb{K}$ and $\mathbb{L}$ that are not in $\mathcal{CR}(\mathbb{I}, \mathbb{J})$.

in $\mathbb{D}$. Additionally, it does not make sense to formulate a distinguishing property involving a non-shared action, as one system has no knowledge of the action (e.g., $\langle c\rangle_o\top$ cannot be evaluated in $\mathbb{B}$). Therefore, to characterize consistency over different vocabularies, we must establish that M and N are consistent if and only if $M@(\alpha M \cap \alpha N)$ and $N@(\alpha M \cap \alpha N)$ are consistent. By Theorem 6, it follows that M and N are consistent if and only if no $\mathcal{L}_\mu^w$ property distinguishes $M@(\alpha M \cap \alpha N)$ and $N@(\alpha M \cap \alpha N)$. We separate the soundness and completeness of this characterization into Theorems 7 and 8.

Theorem 7. (Consistency Characterization (Soundness): Different Vocabularies) *If M and N are consistent MTSs, then $M@(\alpha M \cap \alpha N)$ and $N@(\alpha M \cap \alpha N)$ are consistent.*

Proof:

If $Q \in \mathcal{CR}(M, N)$, then $Q \in \mathcal{CR}(M@X, N@X)$ by Lemma 2 in Section 5.1, where $X = \alpha M \cap \alpha N$. Hence, $M@X$ and $N@X$ are consistent. □

The other direction of Theorem 7 does not hold in general. For example, consider models $\mathbb{I}$ and $\mathbb{J}$ in Figure 4.2 with $\mathtt{c} \notin \alpha\mathbb{J}$ and $\mathtt{d} \notin \alpha\mathbb{I}$. There is no common refinement of $\mathbb{I}$ and $\mathbb{J}$. Intuitively, any P in $\mathcal{CR}(\mathbb{I}, \mathbb{J})$ must satisfy $P \overset{c}{\Longrightarrow}_r P'$ for some P' that proscribes $\mathtt{b}$; otherwise, it would not refine $\mathbb{I}$. Therefore, $P@\alpha\mathbb{J} \overset{\epsilon}{\Longrightarrow}_r P'$ for some P' that proscribes $\mathtt{b}$, whereas $\mathbb{J}$ can still transition on $\mathtt{b}$ after every transition on ϵ. This violates the conditions for refinement in Definition 5, and therefore no common refinement exists. On the other hand, $\mathbb{I}@(\alpha\mathbb{I} \cap \alpha\mathbb{J})$ and $\mathbb{J}@(\alpha\mathbb{I} \cap \alpha\mathbb{J})$ are equal, and therefore consistent.

In the previous example, the inconsistency between models $\mathbb{I}$ and $\mathbb{J}$ is caused by the fact that transitions on $\mathtt{b}$ in $\mathbb{I}$ are only proscribed after following a transition on $\mathtt{c}$ (unobservable to $\mathbb{J}$), whereas transitions on $\mathtt{b}$ in $\mathbb{J}$ are only proscribed after following a transition on $\mathtt{d}$ (unobservable to $\mathbb{I}$). Since $\mathtt{c} \neq \mathtt{d}$, these are clearly inconsistent policies. Furthermore, if $\mathbb{I}$ transits on $\mathtt{c}$, the resulting system $\mathbb{I}_1$ is inconsistent with $\mathbb{J}$, and if $\mathbb{J}$ transits on $\mathtt{d}$, the resulting system $\mathbb{J}_1$ is inconsistent with $\mathbb{I}$. Note that models $\mathbb{K}$ and $\mathbb{L}$ are incorrect attempts at building a common refinement of models $\mathbb{I}$ and $\mathbb{J}$, since upon restricting to the vocabulary $\{\mathtt{b}, \mathtt{c}\}$, both $\mathbb{K}$ and $\mathbb{L}$ can internally transit to a state that proscribes $\mathtt{b}$, and hence neither refines $\mathbb{I}$ (and similarly, neither refines $\mathbb{J}$).

It is also possible that a single non-shared action is the source of an inconsistency between two models. For example, consider models $\mathcal{R}$ and $\mathcal{T}$ in Figure 4.1 with $\alpha\mathcal{R} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $\alpha\mathcal{T} = \{\mathtt{a}, \mathtt{b}, \mathtt{d}\}$. There is no common refinement of these models since there is no consistency relation between them. On the other hand, $\mathcal{T}@(\alpha\mathcal{R} \cap \alpha\mathcal{T})$ is refined by $\mathcal{R}@(\alpha\mathcal{R} \cap \alpha\mathcal{T})$ with the refinement relation $\{(\mathcal{T}_0, \mathcal{R}_0), (\mathcal{T}_2, \mathcal{R}_1)\}$, and therefore they are consistent. Intuitively, the inconsistency between $\mathcal{R}$ and $\mathcal{T}$ is caused by the fact that action $\mathtt{d}$ (unobservable to $\mathcal{R}$) must occur before $\mathcal{T}$ can transition on $\mathtt{a}$, yet all successors on $\mathtt{d}$ in $\mathcal{T}$ are inconsistent with the starting point of the required transition on $\mathtt{a}$ in $\mathcal{R}$. Hence, no common refinement of these models exists. On the other hand, the source of the inconsistency is removed when the alphabet is restricted, since $\mathtt{d}$ is replaced by τ.

Theorem 8. (Consistency Characterization (Completeness): Different Vocabularies) *Let M and N be MTSs and let $X = \alpha M \cap \alpha N$. Suppose that $M@X$ and $N@X$ are consistent and let $C_{M@XN@X}$ be the largest consistency relation between them. If the following conditions hold for all $(M_m@X, N_n@X) \in C_{M@XN@X}$, then M and N are consistent:*

1. $M_m \overset{\ell}{\Longrightarrow}_p M_{m'} \wedge \ell \in \alpha M \setminus \alpha N \Rightarrow (M_{m'}@X, N_n@X) \in C_{M@XN@X}$,

2. $N_n \overset{\ell}{\Longrightarrow}_p N_{n'} \wedge \ell \in \alpha N \setminus \alpha M \Rightarrow (M_m@X, N_{n'}@X) \in C_{M@XN@X}$.

Proof:

Define the set C_{MN} as follows:

$$C_{MN} = \{(M_m, N_n) \mid (M_m@X, N_n@X) \in C_{M@XN@X}\}.$$

We show that C_{MN} is a consistency relation between M and N. Clearly $(M, N) \in C_{MN}$. Let $(M_m, N_n) \in C_{MN}$.

$$M_m \overset{\hat{\ell}}{\Longrightarrow}_r M_{m'} \wedge \ell \in X \cup \{\tau\} \wedge (M_m, N_n) \in C_{MN}$$

($\ell \in X \cup \{\tau\}$ and Definition of C_{MN})

$$\Rightarrow \quad M_m@X \overset{\hat{\ell}}{\Longrightarrow}_r M_{m'}@X \wedge \ell \in X \cup \{\tau\} \wedge (M_m@X, N_n@X) \in C_{M@XN@X}$$

(Definition 17)

$$\Rightarrow \quad \exists N_{n'} \cdot N_n@X \overset{\hat{\ell}}{\Longrightarrow}_p N_{n'}@X \wedge (M_{m'}@X, N_{n'}@X) \in C_{M@XN@X}$$

(Properties of @, Condition (2), and Definition of C_{MN})

$$\Rightarrow \quad \exists N_{n'} \cdot \exists x_1, \ldots, x_i, y_1, \ldots, y_j \in Act_\tau \setminus \alpha M \cdot N_n \overset{\hat{x_1}}{\Longrightarrow}_p N_{x_1} \overset{\hat{x_2}}{\Longrightarrow}_p \cdots \overset{\hat{x_i}}{\Longrightarrow}_p N_{x_i} \wedge$$

$$N_{x_i} \overset{\hat{\ell}}{\Longrightarrow}_p N_{y_1} \overset{\hat{y_1}}{\Longrightarrow}_p \cdots \overset{\hat{y_j}}{\Longrightarrow}_p N_{n'} \wedge \forall k \cdot (M_m, N_{x_k}) \in C_{MN} \wedge$$

$$\forall k \cdot (M_{m'}, N_{y_k}) \in C_{MN} \wedge (M_{m'}, N_{n'}) \in C_{MN}$$

$$M_m \overset{\ell}{\Longrightarrow}_r M_{m'} \wedge \ell \in \alpha M \setminus \alpha N \wedge (M_m, N_n) \in C_{MN}$$

(Definition of C_{MN})

$$\Rightarrow \quad M_m \overset{\ell}{\Longrightarrow}_r M_{m'} \wedge \ell \in \alpha M \setminus \alpha N \wedge (M_m@X, N_n@X) \in C_{M@XN@X}$$

(Condition (2))

$$\Rightarrow \quad (M_{m'}@X, N_n@X) \in C_{M@XN@X}$$

(Let $N_{n'} = N_n$ and Definition of C_{MN})

$$\Rightarrow \quad \exists N_{n'} \cdot N_n \overset{\epsilon}{\Longrightarrow}_p N_{n'} \wedge (M_{m'}, N_{n'}) \in C_{MN}.$$

Hence, Conditions (1) and (3) in Definition 17 hold for C_{MN}, and Conditions (2) and (4) are shown similarly. Therefore, M and N are consistent by Theorem 2. □

We have seen that for models over the same vocabulary, consistency characterization with respect to $\mathcal{L}_\mu^w$ is as expected: the two systems cannot disagree on a property. For models over different vocabularies, however, many subtle issues arise with respect to non-shared actions. In particular, distinguishing properties are only meaningful for equal vocabularies, and therefore the systems must be restricted to their shared alphabet. In addition, we have seen that inconsistencies can be ultimately caused by non-shared actions, which are removed upon reducing to the shared alphabet. Therefore, some of the allowable behaviours on these actions must be restricted in order to characterize consistency in this context.

Because FLTL is preserved by, but does not characterize refinement (see discussion after Theorem 5), it does not characterize consistency. In particular, if two MTSs over the same vocabulary are consistent, no FLTL property distinguishes them (i.e., $\Rightarrow$ of Theorem 6 holds for FLTL), but there is not necessarily an FLTL property that distinguishes between inconsistent systems (i.e., $\Leftarrow$ of Theorem 6 does not hold for FLTL). In addition, if an FLTL property distinguishes two MTSs restricted to their shared alphabet, then the original systems are inconsistent, but the other direction does not hold (i.e., Theorem 8 does not hold for FLTL).

Chapter 5

Algebraic Properties

In practical applications, there may be several system components and relationships between them (e.g., through refinement or other compositional operators). In order for merge to applied in such cases, the merge operation must satisfy certain properties. For example, if we elaborate the merge operands, will the merge of the elaborated views preserve the properties of the original merge? Does the order of merge matter? What is the relationship to parallel composition? In this chapter, we study such *algebraic* properties of the $+$ operator, defined in Section 3.2, both for the case when the LCR of the operands exists, and when multiple non-equivalent MCRs exist. In the former case, we show that most of the desired properties hold (Section 5.1). In the latter case, the existence of multiple non-equivalent MCRs impacts the desired properties, but we show that the right choices of merge can be made in order to guarantee a particular algebraic property (Section 5.2). These results are applied to the case study described in Chapter 7.

5.1 Properties of Least Common Refinements

Throughout this section, whenever we write $M + N$, it is assumed that M and N are consistent MTSs and $+$ results in $\mathcal{LCR}_{M,N}$. Note that by Theorem 3, the properties in this section are guaranteed to hold when all merge operands satisfy the non-shared condition, the multiple-maybe condition, and the determinacy condition. We begin with a useful lemma.

Lemma 2. $B \subseteq A \Rightarrow (M@A \preceq_o N@A \Rightarrow M@B \preceq_o N@B)$.

That is, refinement is preserved by alphabet reduction.

Proposition 3. *For MTSs M, N, and P, the $+$ operator has the following properties:*

1. *(Idempotency)* $M + M \equiv_o M$.
2. *(Commutativity)* $M + N \equiv_o N + M$.
3. *(Associativity)* $(M + N) + P \equiv_o M + (N + P)$.

Proof:

(1) and (2) follow from Definition 12. (3) Let Q be $\mathcal{LCR}_{M+N,P}$.

$$N + P \preceq_o Q@(\alpha N \cup \alpha P)$$

(Definition 12)

$$\Leftarrow \quad Q@(\alpha N \cup \alpha P) \in \mathcal{CR}(N, P)$$

(Definition 11)

$$\Leftarrow \quad N \preceq_o Q@\alpha N \wedge P \preceq_o Q@\alpha P$$

($N \preceq_o (M + N)@\alpha N \preceq_o Q@\alpha N$ by Lemma 2)

$$\Leftarrow \quad N \preceq_o (M + N)@\alpha N \wedge (M + N) \preceq_o Q@(\alpha M \cup \alpha N) \wedge P \preceq_o Q@\alpha P$$

(Definition 15 and Definition 12)

$$= \quad \mathbf{t}.$$

Since, $M \preceq_o (M + N)@\alpha M \preceq_o Q@\alpha M$ by Lemma 2, it follows that $M + (N + P) \preceq_o Q$ by Definition 12. The other direction is proven similarly. □

A useful property of $+$ is monotonicity with respect to observational refinement:

$$(M \preceq_o P) \wedge (N \preceq_o Q) \Rightarrow M + N \preceq_o P + Q. \tag{5.1}$$

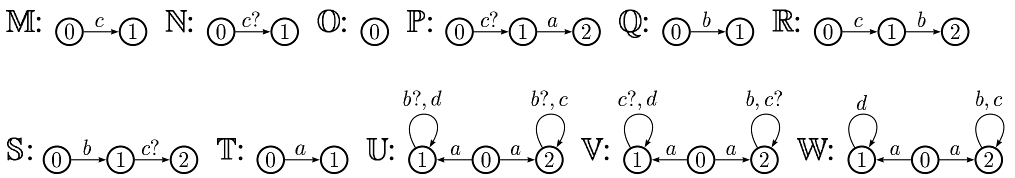

Figure 5.1: Example MTSs for algebraic properties.

This allows for elaborating different viewpoints independently while ensuring that the properties of the original viewpoints put together still hold.

Proposition 4. (Monotonicity) *The operator $+$ is monotonic with respect to observational refinement, i.e., Equation (5.1) holds.*

Proof:

$$(M \preceq_o P) \wedge (N \preceq_o Q) \Rightarrow M + N \preceq_o P + Q$$

(Definition 12)

$$\Leftarrow \quad (M \preceq_o P) \wedge (N \preceq_o Q) \Rightarrow P + Q \in \mathcal{CR}(M, N)$$

(Properties of $\Rightarrow$ and $\wedge$, and Definition 11)

$$\Leftarrow \quad (M \preceq_o P \Rightarrow M \preceq_o (P + Q)@\alpha M) \wedge (N \preceq_o Q \Rightarrow N \preceq_o (P + Q)@\alpha N)$$

(Since $P \preceq_o (P + Q)@\alpha P$, $Q \preceq_o (P + Q)@\alpha Q$, $\alpha P = \alpha M$, and $\alpha Q = \alpha N$)

$$= \quad \mathbf{t}.$$

□

We now look at distributing a parallel composition over merging. Assume that two stakeholders have developed partial models M and N of the intended behaviour of a component M. Each stakeholder will have verified that some required properties hold in a given context (other components and assumptions on the environment P_1, . . ., P_n). It would be desirable if merging viewpoints M and N preserved the properties of both stakeholders under the same assumptions on the environment, i.e., in $(M + N) \parallel P_1 \parallel \cdots \parallel P_n$. This reasoning would be supported if:

$$(M \| P_1 \| \cdots \| P_n) + (N \| P_1 \| \cdots \| P_n) \preceq_o (M + N) \| P_1 \| \cdots \| P_n. \tag{5.2}$$

Unfortunately, Equation (5.2) does not hold in general, unless some restrictions are imposed on the model vocabularies.

Example 1. *Consider models* $\mathbb{M}$*,* $\mathbb{N}$*, and* $\mathbb{O}$ *in Figure 5.1 and assume that* $\alpha\mathbb{O} = \emptyset$*.* $\mathbb{N} + \mathbb{O}$ *is always equivalent to* $\mathbb{N}$*, and by rule MT in Figure 2.3, so is* $(\mathbb{N} + \mathbb{O})||\mathbb{M}$*. On the other hand,* $\mathbb{N}||\mathbb{M}$ *is equal to* $\mathbb{N}$ *and* $\mathbb{O}||\mathbb{M}$ *is equal to* $\mathbb{M}$*, by rules MT and TD, respectively. It follows that* $(\mathbb{O}||\mathbb{M})+(\mathbb{N}||\mathbb{M})$ *is equivalent to* $\mathcal{M}$*, and hence:* $(\mathbb{O}||\mathbb{M})+(\mathbb{N}||\mathbb{M}) \equiv_o \mathbb{M} \not\preceq_o \mathbb{N} \equiv_o (\mathbb{O}+\mathbb{N})||\mathbb{M}$*.*

The desired property fails due to the parallel composition of $\mathbb{O}$ and $\mathbb{M}$. Since c does not belong to $\alpha\mathbb{O}$, parallel composition does not restrict the occurrence of c when composing $\mathbb{O}$ with $\mathbb{M}$. However, this is methodologically wrong if we assume that $\mathbb{O}$ and $\mathbb{N}$ model the same component (which is reasonable because $\mathbb{O}$ and $\mathbb{N}$ are being merged). From $\mathbb{N}$, we know that the system modelled by $\mathbb{O}$ can communicate over c. Hence, c should be included in $\alpha\mathbb{O}$; otherwise, the communicating interface between the components modelled by $\mathbb{O}$ and $\mathbb{M}$ is under-specified. Therefore, when composing two partial models in parallel, the stakeholders must include the full alphabet of the component they are modelling in the alphabet of their partial descriptions, i.e., we must assume that $\alpha P \subseteq \alpha M \cap \alpha N$, where $P = P_1\|\cdots\|P_n$, for the desired property to hold.

Proposition 5. (Distributivity) *If* M*,* N*, and* P *are MTSs such that* $\alpha P \subseteq \alpha M \cap \alpha M$*, then* $(M\|P) + (N\|P) \preceq_o (M + N)\|P$*.*

Proof:

$$(M\|P) + (N\|P) \preceq_o (M + N)\|P$$

(Definition 12)

$$\Leftarrow \quad (M + N)\|P \in \mathcal{CR}(M\|P, N\|P)$$

(Definition 11)

$$\Leftarrow \quad (M\|P \preceq_o ((M + N)\|P)@\alpha M) \wedge (N\|P \preceq_o ((M + N)\|P)@\alpha N)$$

(Since $\alpha P \subseteq \alpha M \cap \alpha N$)

$$\Leftarrow \quad (M\|P \preceq_o (M + N)@\alpha M\|P) \wedge (N\|P \preceq_o (M + N)@\alpha N\|P)$$

$$\begin{array}{ll} & \text{(Proposition 1)} \\ \Leftarrow & (M \preceq_o (M+N)@\alpha M) \wedge (N \preceq_o (M+N)@\alpha N) \\ & \text{(Definition 15)} \\ = & \mathtt{t}. \end{array}$$

□

Remark 4. *If M and N are consistent, then $M \| P$ and $N \| P$ are consistent by monotonicity of parallel composition (Proposition 1) and the fact that $\alpha P \subseteq \alpha M \cap \alpha N$.*

The other direction of Proposition 5 does not hold:

$$(M+N)\|P \not\preceq_o (M\|P) + (N\|P).$$

Intuitively, the composition of M with P may restrict the behaviours of M, for instance, making certain states of M unreachable. It is possible that $M\|P + N\|P$ does not refine $(M+N)\|P$ because inconsistencies are caused by those states of M that are unreachable in $M\|P$.

Example 2. *Assume that models $\mathbb{N}$, $\mathbb{P}$, and $\mathbb{O}$ in Figure 5.1 are over the vocabulary $\{\mathtt{a}, \mathtt{c}\}$. Models $\mathbb{N}$ and $\mathbb{P}$ are consistent and their LCR is $\mathbb{O}$. So, $(\mathbb{N}+\mathbb{P})\|\mathbb{N} \equiv_o \mathbb{O}$ by the rules in Figure 2.3. On the other hand, $\mathbb{N}\|\mathbb{N} = \mathbb{N}$ and $\mathbb{P}\|\mathbb{N} = \mathbb{N}$, and therefore by Idempotency, $\mathbb{N}\|\mathbb{N} + \mathbb{P}\|\mathbb{N} \equiv_o \mathbb{N}$. Since $\mathbb{O} \not\preceq_o \mathbb{N}$, the result follows.*

In Example 2, $\mathbb{N}$ and $\mathbb{P}$ have a disagreement on $\mathtt{a}$ after following the maybe transitions on $\mathtt{c}$, which results in the merge $\mathbb{O}$. The source of this disagreement is removed upon composing both $\mathbb{N}$ and $\mathbb{P}$ with $\mathbb{N}$, because $\mathbb{N}$ restricts the behaviour of $\mathbb{P}$ on $\mathtt{a}$. The merge of $\mathbb{N}\|\mathbb{N}$ and $\mathbb{P}\|\mathbb{N}$ therefore allows more behaviours, and does not refine $(\mathbb{N}+\mathbb{P})\|\mathbb{N}$.

5.2 Properties of Minimal Common Refinements

In this section, we present algebraic properties of $+$ without assuming the existence of the LCR. The algebraic properties are therefore stated in terms of sets and the different choices that can be made when picking an MCR. Idempotence is the only property in Section 5.1 that still holds, since an LCR always exists between a system and itself. The rest of the properties discussed in Section 5.1 require some form of weakening.

Commutativity does not hold in general: if M and N are any two MTSs that have at least two different MCRs, then certainly not every $M + N$ is equivalent to every $N + M$. On the other hand, $\mathcal{MCR}(M, N)$ is always equal to $\mathcal{MCR}(N, M)$, and therefore the same MCR can be chosen.

Proposition 6. (Commutativity) $\mathcal{MCR}(M, N) = \mathcal{MCR}(N, M)$.

Associativity fails for the same reason that commutativity fails. The strongest form of associativity in terms of sets is:

$$\forall A \in \mathcal{MCR}(M, N) \cdot \forall B \in \mathcal{MCR}(N, P) \cdot \mathcal{MCR}(A, P) = \mathcal{MCR}(M, B). \quad (5.3)$$

The following example shows that Equation (5.3) does not hold in general.

Example 3. *Consider models* $\mathbb{M}$*,* $\mathbb{Q}$*, and* $\mathbb{T}$ *in Figure 5.1 and assume that* $\alpha\mathbb{M} = \{\mathtt{c}\}$*,* $\alpha\mathbb{Q} = \{\mathtt{b}\}$*, and* $\alpha\mathbb{T} = \{a\}$*. Model* $\mathbb{R}$ *is in* $\mathcal{MCR}(\mathbb{M}, \mathbb{Q})$*, and there is no* $D \in \mathcal{MCR}(\mathbb{T}, \mathbb{M})$ *such that* $\mathcal{MCR}(\mathbb{R}, \mathbb{T}) = \mathcal{MCR}(\mathbb{Q}, D)$*.*

In Example 3, $\mathbb{R}$ requires that action $\mathtt{c}$ precedes action $\mathtt{b}$ in every trace, and therefore so does every MCR of $\mathbb{R}$ and $\mathbb{T}$, since neither $\mathtt{b}$ nor $\mathtt{c}$ is in $\alpha\mathbb{T}$. However, because $\mathtt{b}$ is not in $\alpha\mathbb{T}$, $\alpha\mathbb{M}$, or $\alpha\mathbb{Q}$, for every D in $\mathcal{MCR}(\mathbb{T}, \mathbb{M})$, there is an MCR of $\mathbb{Q}$ and D such that action $\mathtt{c}$ follows action $\mathtt{b}$. Hence, $\mathcal{MCR}(\mathbb{R}, \mathbb{T}) \neq \mathcal{MCR}(\mathbb{Q}, D)$ for *any* D in $\mathcal{MCR}(\mathbb{T}, \mathbb{M})$. In fact, Example 3 shows that there exists M, N, and P such that:

$$\exists A \in \mathcal{MCR}(M, N) \cdot \forall B \in \mathcal{MCR}(N, P) \cdot \mathcal{MCR}(A, P) \neq \mathcal{MCR}(M, B).$$

Therefore, set equality of $\mathcal{MCR}(A, P)$ and $\mathcal{MCR}(M, B)$ for associativity is not possible.

A weaker form of associativity in terms of sets is:

$$\begin{aligned} &\forall A \in \mathcal{MCR}(M, N) \cdot \forall B \in \mathcal{MCR}(N, P) \cdot (\mathcal{CR}(A, P) \neq \emptyset \wedge \mathcal{CR}(M, B) \neq \emptyset) \quad (5.4) \\ \Rightarrow \quad &\mathcal{MCR}(A, P) \cap \mathcal{MCR}(M, B) \neq \emptyset. \end{aligned}$$

This form does not require that $\mathcal{MCR}(A,P)$ and $\mathcal{MCR}(M,B)$ be equal, but rather that they share an MCR (up to equivalence) if the models are consistent. Unfortunately, Equation (5.4) does not hold either, illustrated by the following example.

Example 4. *Consider models $\mathbb{N}$ and $\mathbb{Q}$ in Figure 5.1 with $\alpha\mathbb{N} = \{\mathtt{c}\}$ and $\alpha\mathbb{Q} = \{\mathtt{b}\}$. We show that $\mathbb{N} + (\mathbb{Q} + \mathbb{N}) \not\equiv_o (\mathbb{N} + \mathbb{Q}) + \mathbb{N}$ for $\mathbb{R}$ and $\mathbb{S}$ in $\mathcal{MCR}(\mathbb{N}, \mathbb{Q})$. $\mathcal{LCR}_{\mathbb{R},\mathbb{N}}$ is $\mathbb{R}$, and $\mathcal{LCR}_{\mathbb{N},\mathbb{S}}$ is $\mathbb{S}$. Because $\mathbb{R} \not\equiv_o \mathbb{S}$, every MCR of $\mathbb{R}$ and $\mathbb{N}$ is incomparable with every MCR of $\mathbb{N}$ and $\mathbb{S}$, and so the desired form of associativity fails.*

Examples 3 and 4 show that neither set equality nor fixing both A in $\mathcal{MCR}(M,N)$ and B in $\mathcal{MCR}(N,P)$ are possible for associativity. Instead, the following proposition outlines two forms of associativity, without set equality, that fix some A in $\mathcal{MCR}(M,N)$ or some B in $\mathcal{MCR}(N,P)$, but not both.

Proposition 7. (Associativity) *If M, N, and P are MTSs, then:*

1. $\forall A \in \mathcal{MCR}(M,N) \cdot \exists B \in \mathcal{MCR}(N,P) \cdot (\mathcal{MCR}(A,P) \cap \mathcal{MCR}(M,B) \neq \emptyset)$*, and*
2. $\forall B \in \mathcal{MCR}(N,P) \cdot \exists A \in \mathcal{MCR}(N,P) \cdot (\mathcal{MCR}(A,P) \cap \mathcal{MCR}(M,B) \neq \emptyset)$*,*

where $\mathcal{CR}(A,P) \neq \emptyset$ and $\mathcal{CR}(M,B) \neq \emptyset$.

Proof:

(Sketch) (1) Let $A \in \mathcal{MCR}(M,N)$ and $Q \in \mathcal{MCR}(A,P)$. By Definition 14, $N \preceq_o A@\alpha N$ and $A \preceq_o Q@\alpha A = Q@(\alpha M \cup \alpha N)$. Hence, by Lemma 2, $N \preceq_o A@\alpha N \preceq_o Q@\alpha N$, and similarly it follows that $P \preceq_o Q@\alpha P$. Since $Q@(\alpha N \cup \alpha P)$ is in $\mathcal{CR}(Q@\alpha N, Q@\alpha P)$ by Lemma 2, there exists B in $\mathcal{MCR}(N,P)$ such that $B \preceq_o Q@(\alpha N \cup \alpha P)$ (every CR refines some MCR). Hence, Q is in $\mathcal{CR}(M,B)$. Therefore, $\mathcal{CR}(A,P) \cap \mathcal{CR}(M,B)$ is non-empty, which implies that $\mathcal{MCR}(A,P) \cap \mathcal{MCR}(M,B)$ is non-empty. (2) is proven similarly. □

Condition (1) in Proposition 7 says that for any $M + N$, there exists some $N + P$ such that the same MCR for $(M + N) + P$ and $M + (N + P)$ can be selected. Condition (2) is analogous to condition (1) with the roles of $M + N$ and $N + P$ reversed. Note that Proposition 7 reduces to Proposition 3 if all sets of MCRs are singletons (up to equivalence).

Monotonicity is also disrupted by multiple MCRs. It is not expected that any choice of $M + N$ is refined by any choice of $P + N$ when M is refined by P, because incompatible decisions may be made in the two merges. Rather, there are two desirable forms of monotonicity: (1) whenever $M + N$ is chosen, some $P + N$ can be chosen such that $P + N$ refines $M + N$; and (2) whenever $P + N$ is chosen, then some $M + N$ can be chosen such that $P + N$ refines $M + N$. Form (1) does not hold, as the following example shows.

Example 5. *Models $\mathbb{N}$ and $\mathbb{Q}$ in Figure 5.1 with $\alpha\mathbb{N} = \{\mathtt{c}\}$ and $\alpha\mathbb{Q} = \{\mathtt{b}\}$ are consistent, and their merge $\mathbb{N} + \mathbb{Q}$ may result in model $\mathbb{R}$. Also, $\mathbb{N} \preceq_o \mathbb{O}$ (assuming that $\alpha\mathbb{O} = \{\mathtt{c}\}$) and models $\mathbb{O}$ and $\mathbb{Q}$ are consistent. However, $\mathcal{LCR}_{\mathbb{O},\mathbb{Q}}$ is equivalent to $\mathbb{Q}$ over $\{\mathtt{b}, \mathtt{c}\}$, and since $\mathbb{Q} \not\preceq_o \mathbb{R}$, no MCR of $\mathbb{O}$ and $\mathbb{Q}$ that refines $\mathbb{R}$ can be chosen.*

Form (1) fails because there are two choices of refinement being made. On the one hand, by picking a minimal common refinement for M and N over others, we are deciding over incompatible refinement choices. On the other hand, we are choosing how to refine M into P. These two choices need not be consistent, leading to failure of monotonicity. This tells us that choosing an MCR adds information to the merged model, which may be inconsistent with evolutions of the different viewpoints that are represented by the models being merged. Form (2) always holds, as stated below.

Proposition 8. (Monotonicity) *If M, N, P, and Q are MTSs, then:*

$$M \preceq_o P \wedge N \preceq_o Q \Rightarrow \forall B \in \mathcal{MCR}(P, Q) \cdot \exists A \in \mathcal{MCR}(M, N) \cdot A \preceq_o B$$

Proof:

$P + Q$ is always in $\mathcal{CR}(M, N)$, and thus refines some MCR of M and N. □

Thus, once $P + Q$ is chosen, there always exists some $M + N$ that it refines, and so the properties of M and N are preserved in $P + Q$. Note that if $\mathcal{MCR}(M, N)$ is a singleton set, Proposition 8 reduces to Proposition 4, as expected. In practical terms, this means that if the various viewpoints are still to be elaborated, the results of reasoning about one of their possible merges (picked arbitrarily) are not guaranteed to carry through once the viewpoints have been further refined.

We now address distributivity in the context of multiple MCRs. Similar to monotonicity, there are two desirable forms of this property: (1) given any A in $\mathcal{MCR}(M\|P, N\|P)$, there is some B in $\mathcal{MCR}(M, N)$ such that A is refined by $B\|P$; and (2) given any B in $\mathcal{MCR}(M, N)$, there is some A in $\mathcal{MCR}(M\|P, N\|P)$ such that A is refined by $B\|P$. Form (2) does not hold, as the following example shows.

Example 6. *Consider models $\mathbb{O}$, $\mathbb{U}$, $\mathbb{V}$, and $\mathbb{W}$ in Figure 5.1 and assume that $\alpha\mathbb{O} = \{d\}$, and $\mathbb{U} = \mathbb{V} = \mathbb{W} = \{a, b, c, d\}$. Additionally, consider models $\mathcal{W}$, $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ in Figure 3.2 over the vocabulary $\{a, b, c, d\}$. By the rules in Figure 2.3, $\mathbb{U}\|\mathbb{O} = \mathcal{W}$ and $\mathbb{V}\|\mathbb{O} = \mathcal{X}$. Furthermore, recall from Section 4.2 that $\mathcal{Z} \in \mathcal{MCR}(\mathcal{W}, \mathcal{X})$. On the other hand, $\mathcal{LCR}_{\mathbb{U},\mathbb{V}}$ is $\mathbb{W}$, and $\mathbb{W}\|\mathbb{O} = \mathcal{Y}$, which is not a refinement of $\mathcal{Z}$.*

In the previous example, $\mathcal{LCR}_{\mathbb{U},\mathbb{V}}$ exists by Theorem 3 because the required transitions on d in these models restrict the choices that can be made with respect to combining the non-determinism on action a: ($\mathbb{U}_1$ and $\mathbb{V}_1$) and ($\mathbb{U}_2$ and $\mathbb{V}_2$) are consistent, but neither ($\mathbb{U}_1$ and $\mathbb{V}_2$) nor ($\mathbb{U}_2$ and $\mathbb{V}_1$) are consistent. Upon composing with $\mathbb{O}$, the source of the inconsistencies between ($\mathbb{U}_1$ and $\mathbb{V}_2$) and ($\mathbb{U}_2$ and $\mathbb{V}_1$) is removed, and consequently, $\mathcal{LCR}_{\mathbb{U}\|\mathbb{O},\mathbb{V}\|\mathbb{O}}$ does not exist. In particular, similar to Example 2 in Section 5.1, parallel composition may remove inconsistencies between M and N, allowing for more common refinements of $M\|P$ and $N\|P$.

On the other hand, Form (2) holds, and is of particular utility when elaborating models from different viewpoints, as we show in the case study in Chapter 7.

Proposition 9. (Distributivity) *If M, N, and P are such that $\alpha P \subseteq \alpha M \cap \alpha N$, then:*

$$\forall B \in \mathcal{MCR}(M, N) \cdot \exists A \in \mathcal{MCR}(M\|P, N\|P) \cdot A \preceq_o B\|P.$$

Proof:

Analogous to Proposition 8. □

In this section, we have shown that when $+$ does not necessarily produce an LCR, most properties studied in Section 5.1 fail to hold. Intuitively, the existence of inequivalent MCRs implies that merging involves a choice that requires some form of human intervention: a choice which is loaded with domain knowledge. This impacts the results on algebraic properties when moving from LCRs to MCRs. However, we have shown that the right choices of MCRs can usually be made in order to guarantee a particular algebraic property, if desired.

Chapter 6

Algorithms

In this chapter, we describe algorithms related to constructing merge that are intended to support the process outlined in Figure 6.1. We begin by motivating the use of observation graphs for constructing merge (Section 6.1). In the subsequent sections, we describe four algorithms: Algorithm 1 for checking consistency and computing the largest consistency relation (Section 6.2), Algorithm 2 for building an $\mathcal{L}_\mu^w$ property that distinguishes between inconsistent systems (Section 6.3), the $+_{cr}$ operator for building a common refinement (Section 6.4), and the $+_{ca}$ operator for building a common abstraction of all MCRs (Section 6.5). Finally, Section 6.6 discusses the results obtained in this section and their relationship to the desired merge process in Figure 6.1.

6.1 Using Observation Graphs to Construct Merge

MTSs are equipped with two transition relations that describe simple transitions (i.e., on $\longrightarrow$). On the other hand, we are interested in building a common *observational* refinement, which preserves observable transitions (i.e., on $\Longrightarrow$). In particular, to build a merge of two MTSs, it may be necessary to combine observable transitions that pass through states that must be omitted from the merge because they are the source of an inconsistency. For example, models A and B in Figure 6.3 are consistent with consistency relation $C_{\mathsf{AB}} = \{(\mathsf{A}_0, \mathsf{B}_0), (\mathsf{A}_2, \mathsf{B}_1), (\mathsf{A}_3, \mathsf{B}_2)\}$

MERGE. *(The Merge Process)*

```
Input:  MTSs $M$ and $N$
Output: If $M$ and $N$ are consistent, return the desired merge;
        Otherwise, return a distinguishing property.

If ($M$ and $N$ are consistent (Algorithm 1))
    If (stakeholders do not require minimality)
        return element of $\mathcal{CR}(M, N)$ ($+_{cr}$ operator);
    Else
        If (LCR exists (Theorem 3)) return $\mathcal{LCR}_{M,N}$ ($+_{cr}$ operator);
        Else return $\mathcal{GCA}_{M,N}$ ($+_{ca}$ operator);
Else return distinguishing property (Algorithm 2);
```

Figure 6.1: An algorithm for the merge process.

and B is in $\mathcal{CR}(\mathsf{A}, \mathsf{B})$. The transition $\mathsf{B}_0 \xrightarrow{a}_r \mathsf{B}_1$ is matched in C_{AB} with $\mathsf{A}_0 \overset{a}{\Longrightarrow}_m \mathsf{A}_2$, and cannot be matched with $\mathsf{A}_0 \xrightarrow{a}_m \mathsf{A}_1$ because A_1 and B_1 are inconsistent. Thus, in order to merge A and B, we must combine $\mathsf{A}_0 \overset{a}{\Longrightarrow}_m \mathsf{A}_2$ and $\mathsf{B}_0 \xrightarrow{a}_r \mathsf{B}_1$ without including intermediate states in A. Thus, merging these two transitions cannot be done by combining multiple simple transitions, as some lead to inconsistencies. Instead, we must build the transition relations for the merged model from the observable transition relations, which are naturally computed using the observation graphs of the original systems. In particular, the transition $0 \xrightarrow{a}_m 2$ in A_ϵ can be consistently combined with $0 \xrightarrow{a}_r 1$ in B_ϵ (see Figure 6.3).

In addition, consistency relations and distinguishing properties in $\mathcal{L}^w_\mu$ also utilize observable transitions. Hence, through this chapter, our algorithms are stated in terms of observable transitions, which can be obtained by first computing the observation graphs. Doing so for an MTS $M = (S_M, L_M, \Delta^r_M, \Delta^p_M, s_{0M})$ using standard techniques for computing products and transitive closures of relations takes time $O(|L_M| \times |S_M|^3)$ [9].

6.2 Building the Largest Consistency Relation

Ideally, information from the stakeholders should be used to compute a consistency relation. Behaviours that can be combined via a consistency relation C_{MN}, but that do not affect whether

C_{MN} is a consistency relation may not be desirable behaviours of the merge, and may overly complicate it. Instead, we show how to compute the *largest* consistency relation between two systems, which captures all possible areas of agreement. This consistency relation can either be used to construct the merge directly (see Sections 6.4 and 6.5), or can first be modified and subsequently used to construct the merge. For example, $C_{\mathbb{X}\mathbb{Y}} = \{(\mathbb{X}_0,\mathbb{Y}_0),\ (\mathbb{X}_1,\mathbb{Y}_1),\ (\mathbb{X}_1,\mathbb{Y}_2),\ (\mathbb{X}_2,\mathbb{Y}_3),\ (\mathbb{X}_3,\mathbb{Y}_4),\ (\mathbb{X}_4,\mathbb{Y}_5)\}$ is the largest consistency relation between $\mathbb{X}$ and $\mathbb{Y}$ in Figure 6.3. However, if it is known that $(\mathbb{X}_1, \mathbb{Y}_4)$ should not be combined, then $C_{\mathbb{X}\mathbb{Y}} \setminus \{(\mathbb{X}_1, \mathbb{Y}_4)\}$, which is still a consistency relation by Definition 17, can be used instead.

Algorithm 1 (or `Consistency`) in Figure 6.2 computes the largest consistency relation between two MTSs. The algorithm takes three arguments: two MTSs M and N, and a set C_{MN} that is initially empty. We assume that pairs (M, N) have a `status` variable that is initially set to `unvisited`. A possible transition in one model is fixed (Lines 5, 17, 24, and 26) and recursively matched with (a sequence of) consistent possible transitions in the other model (Lines 6 – 13, 18 – 20, 25 and 27), according to the conditions in Definition 17. This process continues until all reachable states that can be combined are identified or until a disagreement that makes the models inconsistent is found (Lines 14 – 16 and 21 – 23). Upon termination, if M and N are consistent, C_{MN} is the largest consistency relation between M and N and Algorithm 1 returns `true`; otherwise, it returns `false`.

For example, consider models $\mathbb{X}$ and $\mathbb{Z}$ in Figure 6.3, and suppose `Consistency`($\mathbb{X}$, $\mathbb{Z}$, $C_{\mathbb{X}\mathbb{Z}}$) is called, where $C_{\mathbb{X}\mathbb{Z}}$ is initially the empty set. The pair $(\mathbb{X}_0, \mathbb{Z}_0)$ is not in $C_{\mathbb{X}\mathbb{Z}}$, is not `beingDetermined`, and has not been `determined`. Hence, Line 5 is reached. Because $\mathbb{X}_0 \xrightarrow{a}_r \mathbb{X}_1$, the For loop on Line 5 considers $\mathbb{X}_1$. Line 7 finds $\mathbb{Z}_1$ such that $\mathbb{Z}_0 \overset{a}{\Longrightarrow}_p \mathbb{Z}_1$, and therefore `Consistency`($\mathbb{X}_1$, $\mathbb{Z}_1$, $C_{\mathbb{X}\mathbb{Z}}$) is called on Line 10. In `Consistency`($\mathbb{X}_1$, $\mathbb{Z}_1$, $C_{\mathbb{X}\mathbb{Z}}$), the For loop on Line 5 considers $\mathbb{X}_2$, since $\mathbb{X}_1 \xrightarrow{b}_m \mathbb{X}_2$. Line 7 searches for matching transitions in $\mathbb{Z}$, but because $\mathbb{Z}_1$ cannot transition on `b`, Line 14 is reached. The set `Matches` is empty, but the transition $\mathbb{X}_1 \xrightarrow{b}_m \mathbb{X}_2$ is not required (i.e., not the source of an inconsistency), and so the algorithm continues. Since $\mathbb{Z}_1$ has no outgoing transitions, the pair $(\mathbb{X}_1, \mathbb{Z}_1)$ is added

ALGORITHM 1. *(Consistency Algorithm)*

Input: MTSs M and N, and a set C_{MN} (initially empty).
Output: if M and N are consistent, returns `true` and C_{MN} is the the largest consistency relation between M and N; otherwise, returns `false`.

```
1:  Consistency($M$, $N$, $C_{MN}$)
2:      If (($M, N$) $\in C_{MN}$ or ($M, N$).status = beingDetermined) return true;
3:      Else If (($M, N$).status = determined) return false;
4:      ($M, N$).status = beingDetermined;
5:      For ($M'$ such that $M \xRightarrow{\hat{\ell}}_p M'$ for some $\ell \in (\alpha M \cap \alpha N) \cup \{\tau\}$)
6:          Matches = $\emptyset$; // the set of matches in $N$ for $M'$
7:          For ($N \xRightarrow{\hat{x_1}}_p N_{x_1} \cdots \xRightarrow{\hat{x_i}}_p N_{x_i} \xRightarrow{\hat{\ell}}_p N_{y_1} \xRightarrow{\hat{y_1}}_p N_{y_2} \cdots \xRightarrow{\hat{y_j}}_p N'$,
                where $x_1, x_2, \ldots, x_i, y_1, y_2, \ldots, y_j \in (Act_\tau \setminus \alpha M)$) // find matches for $M'$
8:              match = true; // keeps track of whether the paths in $M$ and $N$ can be matched
9:              For (($M_i, N_i$) = ($M', N'$), ($M, N_{x_1}$), ..., ($M, N_{x_i}$), ($M', N_{y_1}$), ..., ($M', N_{y_j}$))
10:                 If ($\neg$Consistency($M_i$, $N_i$, $C_{MN}$)) // the paths cannot be matched
11:                     match = false;
12:                     break; // exit For loop on Line 9
13:             If (match) Matches = Matches $\cup \{N'\}$; // the paths match
14:         If (Matches = $\emptyset$ and $M \xRightarrow{\hat{\ell}}_r M'$) // a required transition could not be matched
15:             ($M, N$).status = determined;
16:             return false;
17:     For ($M'$ such that $M \xRightarrow{\ell}_p M'$ for some $\ell \in \alpha M \setminus \alpha N$)
18:         Matches = $\emptyset$;
19:         For ($N'$ such that $N \xRightarrow{\epsilon}_p N'$) // find matches for $M'$
20:             // analogous to Lines 8 – 13
21:         If (Matches = $\emptyset$ and $M \xRightarrow{\ell}_r M'$) // a required transition could not be matched
22:             ($M, N$).status = determined;
23:             return false;
24:     For ($N'$ such that $N \xRightarrow{\hat{\ell}}_p N'$ for some $\ell \in (\alpha M \cap \alpha N) \cup \{\tau\}$)
25:         // analogous to Lines 6 – 16
26:     For ($N'$ such that $N \xRightarrow{\ell}_p N'$ for some $\ell \in \alpha N \setminus \alpha M$)
27:         // analogous to Lines 18 – 23
28:     ($M, N$).status = determined;
29:     $C_{MN}$ = $C_{MN}$ $\cup\{(M, N)\}$;
30:     return true;
```

Figure 6.2: An algorithm for computing the largest consistency relation.

to $C_{\mathbb{X}\mathbb{Z}}$ on Line 29, and `Consistency`($\mathbb{X}_1$, $\mathbb{Z}_1$, $C_{\mathbb{X}\mathbb{Z}}$) returns `true` on Line 30. Hence, back in `Consistency`($\mathbb{X}$, $\mathbb{Z}$, $C_{\mathbb{X}\mathbb{Z}}$), $\mathbb{Z}_1$ is added to `Matches` on Line 13. Next, the For loop on Line 7 considers $\mathbb{Z}_2$, since $\mathbb{Z}_0 \xRightarrow{a}_p \mathbb{Z}_2$, and therefore `Consistency`($\mathbb{X}_1$, $\mathbb{Z}_2$, $C_{\mathbb{X}\mathbb{Z}}$) is called on Line 10. Similarly, the transition $\mathbb{X}_1 \xrightarrow{b}_m \mathbb{X}_2$ is considered on Line 5, but no match in $\mathbb{Z}_2$ is found

(Lines 6 – 13). Again, since the transition on `b` in $\mathbb{X}$ is maybe, the algorithm continues. The required transition $\mathbb{Z}_2 \xrightarrow{c}_r \mathbb{Z}_3$ considered on Line 24 cannot be matched in $\mathbb{X}_1$, and therefore $(\mathbb{X}_1, \mathbb{Z}_2)$ is not a consistent pair and `Consistency`($\mathbb{X}_1$, $\mathbb{Z}_2$, $C_{\mathbb{XZ}}$) returns `false` (analogous to Lines 14 – 16). The set `Matches` in `Consistency`($\mathbb{X}$, $\mathbb{Z}$, $C_{\mathbb{XZ}}$) is already non-empty since it contains $\mathbb{Z}_1$, and so the fact that $\mathbb{X}_1$ and $\mathbb{Z}_2$ are inconsistent does not imply that $\mathbb{X}$ and $\mathbb{Z}$ are inconsistent (i.e., the If condition on Line 14 is not satisfied). Continuing in this way, the consistency relation constructed by Algorithm 1 is $C_{\mathbb{XZ}} = \{(\mathbb{X}_0, \mathbb{Z}_0), (\mathbb{X}_1, \mathbb{Z}_1), (\mathbb{X}_3, \mathbb{Z}_2), (\mathbb{X}_4, \mathbb{Z}_3)\}$, which is indeed the largest consistency relation between $\mathbb{X}$ and $\mathbb{Z}$.

As mentioned above, there is no consistency relation between models $\mathbb{Y}$ and $\mathbb{Z}$ in Figure 6.3. In particular, the required transition $\mathbb{Y}_0 \xrightarrow{a}_r \mathbb{Y}_1$ cannot be consistently matched by $\mathbb{Z}_0 \xrightarrow{a}_r \mathbb{Z}_1$ or $\mathbb{Z}_0 \xrightarrow{a}_r \mathbb{Z}_2$ because $\mathbb{Z}_1 \overset{b}{\not\Longrightarrow}$ and $\mathbb{Z}_2 \overset{b}{\not\Longrightarrow}$. Upon reaching Line 14, the set `Matches` is empty and $\mathbb{Y}_0 \xrightarrow{a}_r$, and thus Algorithm 1 returns `false` on Line 16, as expected.

Theorem 9. (Correctness of Algorithm 1) *Algorithm 1 called with parameters M, N, and $C_{MN} = \emptyset$ returns* `true` *and sets C_{MN} to be the largest consistency relation between M and N if and only if M and N are consistent.*

Proof:

($\Rightarrow$) Suppose C_{MN} is initially empty and M are N are consistent. Upon termination of Algorithm 1, C_{MN} is a consistency relation between M and N: Condition (1) in Definition 17 is satisfied due to Lines 5 – 16, Condition (2) is satisfied due to Lines 24 – 25, Condition (3) is satisfied due to Lines 17 – 23, and Condition (4) is satisfied due to Lines 26 – 27. C_{MN} is the largest consistency relation because Lines 5, 17, 24, and 26 consider possible transitions, and the For loops on Lines 7 and 19 search for all possible matches.

($\Leftarrow$) If M and N are inconsistent, there is no consistency relation between them by Theorem 2, and hence at least one of the conditions in Definition 17 is violated: a required transition in one model cannot be matched in the other. The set `Matches` will remain empty and therefore Algorithm 1 returns `false` (e.g., Lines 16 and 23). □

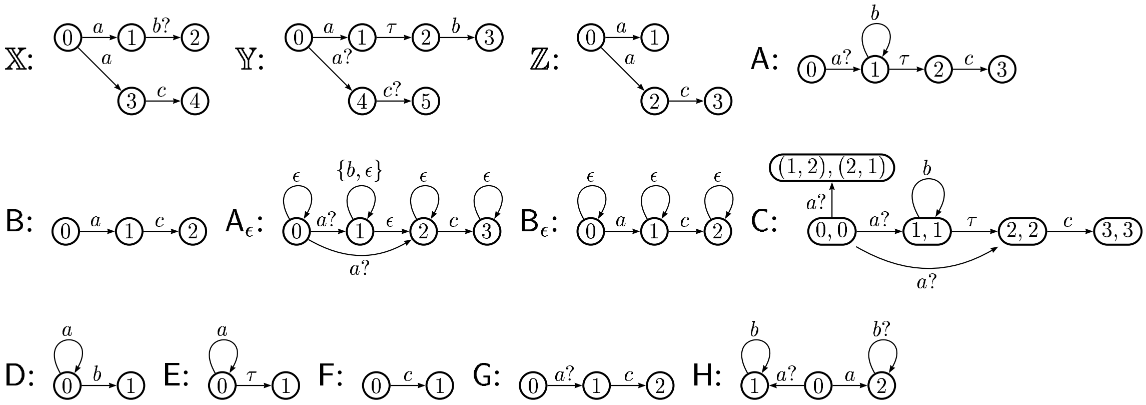

Figure 6.3: Example MTSs for illustrating merge.

We now address complexity issues. Algorithm 1 contains backtracking, and therefore the naive implementation of this algorithm has running time exponential in $|T|$, where $M = (S_M, L_M, \Delta^r_M, \Delta^p_M, s_{0M})$, $N = (S_N, L_N, \Delta^r_N, \Delta^p_N, s_{0M})$, $|T|$ = $|T_M|$ + $|T_N|$, and $|T_i| = |L_i| \times |S_i|^2$ for $i \in \{M, N\}$. However, we believe that it is possible to apply techniques that are similar to those used to efficiently compute bisimulation relations [42] to compute the largest consistency relation between two observation graphs, which takes time $O(|S| \times \log(|T|))$ [44, 13], where $|S| = |S_M| + |S_N|$. We leave for future work improving the complexity of Algorithm 1 using these techniques.

6.3 Finding a Distinguishing Property

If two systems are inconsistent, it is desirable to have a property that gives insight into the specific areas of disagreement. In this section, we give an algorithm for finding such a property for inconsistent systems over the same vocabulary, and show that it can be applied to certain models with different vocabularies.

By Theorem 2, there is no consistency relation between two inconsistent systems. Therefore, if M and N are inconsistent, at least one of the conditions in Definition 17 is violated for the pair (M, N). This gives rise to Algorithm 2 in Figure 6.4 for finding an $\mathcal{L}^w_\mu$ property that distinguishes between inconsistent systems over the same vocabulary. Intuitively, Algorithm 2 traverses simultaneously reachable inconsistent paths until a disagreement is found along all branches, while building a $\mathcal{L}_\mu^w$ property that keeps track of the paths that are traversed. Because there is no consistency relation, following such paths will always lead to a disagreement. Specifically, M and N over the same vocabulary are inconsistent if (without loss of generality) $M \xRightarrow{\hat{\ell}}_r M'$ and either $N \overset{\hat{\ell}}{\not\Longrightarrow}$, or for every N' such that $N \xRightarrow{\hat{\ell}}_p N'$, the MTSs M' and N' are inconsistent. If $N \overset{\hat{\ell}}{\not\Longrightarrow}$, then $\langle\hat{\ell}\rangle_o\top$ distinguishes between M and N; otherwise, $\langle\hat{\ell}\rangle_o(\ldots)$ (in some cases $[\hat{\ell}]_o(\ldots)$) is added into the formula, and the process continues by visiting the pair (M', N'), for every such N'.

For example, suppose that Algorithm 2 is called for inconsistent models $\mathbb{Z}$ and B (both with alphabet $\{\mathsf{a}, \mathsf{c}\}$) in Figure 6.3. Initially, $IS = \{(\mathbb{Z}, \mathsf{B})\}$ (Line 3) and $\phi = \phi_{\mathbb{Z},\mathsf{B}}$ (Line 4). The For loop on Line 6 considers the pair $(\mathbb{Z}, \mathsf{B})$. Since $\mathbb{Z}_0 \xrightarrow{a}_r \mathbb{Z}_1$ and $\mathsf{B}_0 \xrightarrow{a}_r \mathsf{B}_1$ (which is the only transition on a in B), and $\mathbb{Z}_1$ and B_1 are inconsistent on c and are `unvisited`, the If condition on Line 14 is satisfied. Therefore, $\phi_{\mathbb{Z},\mathsf{B}}$ is replaced by $\langle a\rangle_o\phi_{\mathbb{Z}_1,\mathsf{B}_1}$ (Line 16), and IS is set to $\{(\mathbb{Z}_1, \mathsf{B}_1)\}$ (Lines 17 and 22). The For loop on Line 6 then considers $(\mathbb{Z}_1, \mathsf{B}_1)$. Since $\mathsf{B}_1 \xrightarrow{c}_r \mathsf{B}_2$ and $\mathbb{Z}_1 \overset{c}{\not\Longrightarrow}$, the If statement on Line 11 is satisfied. Therefore, $\phi_{\mathbb{Z}_1,\mathsf{B}_1}$ is replaced by $\langle c\rangle_o\bot$ (Line 12) and IS becomes empty (Lines 13 and 22). Therefore, Algorithm 2 returns $\phi = \langle a\rangle_o\langle c\rangle_o\bot$, which is *true* in $\mathbb{Z}$ and *false* in B.

Theorem 10. (Correctness of Algorithm 2) *If M and N are inconsistent MTSs over the same vocabulary, Algorithm 2 called with M and N returns an $\mathcal{L}_\mu^w$ property that distinguishes between them.*

Proof:

Lines 8 – 13 handle disagreements (i.e., one system requires a transition and the other proscribes it), Lines 14 – 17 handle required behaviour in M that cannot be simulated in N, and Lines 18 – 21 handle required behaviour in N that cannot be simulated in M. The While loop (Lines 5 – 22) only exits when all inconsistent paths have been followed to a disagreement. The property returned corresponds to a tree (since Lines 14 and 18 consider `unvisited` pairs) of inconsistent paths between the two models, where the leafs represent disagreements. □

ALGORITHM 2. *(Finding a distinguishing property)*

Input: MTSs $M = (S_M, L_M, \Delta^r_M, \Delta^p_M, s_{0M})$ and $N = (S_N, L_N, \Delta^r_N, \Delta^p_N, s_{0N})$.
Output: if M and N are inconsistent, return $\phi \in \mathcal{L}^w_\mu$ that is *true* in M and *false* in N;
otherwise, return `null`.

```
For ($s \in S_M$ and $t \in S_N$) $(M_s, N_t)$.status = unvisited;
If ($M$ and $N$ are inconsistent)
  $IS = \{(M, N)\}$;
  $\phi = \phi_{M,N}$; // a placeholder for the subformula distinguishing $M$ and $N$
  While $(IS \neq \emptyset)$ // there are more inconsistent pairs to traverse
    For ($(M, N) \in IS$)
      $(M, N)$.status = visited;
      If ($M \stackrel{\ell}{\Longrightarrow}_r M'$ and $N \stackrel{\ell}{\not\Longrightarrow}$) // $\langle \ell \rangle_o \top$ is true in $M$ and false in $N$
        $\phi = \phi[\phi_{M,N} \to \langle \ell \rangle_o \top]$; // substitute $\langle \ell \rangle_o \top$ for $\phi_{M,N}$
        $IS' = IS \setminus \{(M, N)\}$;
      Else If ($N \stackrel{\ell}{\Longrightarrow}_r N'$ and $M \stackrel{\ell}{\not\Longrightarrow}$) // $\langle \ell \rangle_o \bot$ is true in $M$ and false in $N$
        $\phi = \phi[\phi_{M,N} \to \langle \ell \rangle_o \bot]$; // substitute $\langle \ell \rangle_o \bot$ for $\phi_{M,N}$
        $IS' = IS \setminus \{(M, N)\}$;
      Else If (($M \stackrel{\ell}{\Longrightarrow}_r M'$) and ($\forall N' \cdot N \stackrel{\hat{\ell}}{\Longrightarrow}_p N'$ implies $M'$ and $N'$ inconsistent
            and $(M', N')$.status = unvisited))
        // $M'$ is inconsistent with each $N'$; therefore add $(\wedge_{N'_i \in T} \phi_{M',N'_i})$ to $\phi$
        Let $T = \{N' \mid N \stackrel{\hat{\ell}}{\Longrightarrow}_p N'\}$;
        $\phi = \phi[\phi_{M,N} \to \langle \hat{\ell} \rangle_o (\wedge_{N'_i \in T} \phi_{M',N'_i})]$;
        $IS' = (IS \cup \{(M', N'_i) \mid N'_i \in T\}) \setminus \{(M, N)\}$;
      Else If (($N \stackrel{\ell}{\Longrightarrow}_r N'$) and $\forall M' \cdot M \stackrel{\hat{\ell}}{\Longrightarrow}_p M'$ implies $M'$ and $N'$ inconsistent
            and $(M', N')$.status = unvisited))
        // $N'$ is inconsistent with each $M'$; therefore, add $(\vee_{M'_i \in S} \phi_{M'_i,N'})$ to $\phi$
        Let $S = \{M' \mid M \stackrel{\hat{\ell}}{\Longrightarrow}_p M'\}$;
        $\phi = \phi[\phi_{M,N} \to \langle \hat{\ell} \rangle_o (\vee_{M'_i \in S} \phi_{M'_i,N'})]$;
        $IS' = (IS \cup \{(M'_i, N') \mid M'_i \in S\}) \setminus \{(M, N)\}$;
    $IS = IS'$;
  return $\phi$;
Else return null;
```

Figure 6.4: An algorithm for finding a distinguishing property.

Algorithm 2 can also be used for models over different vocabularies. Recall from Section 4.4 that distinguishing properties are defined for models over the shared vocabulary, and, in addition, the models need to satisfy the conditions in Theorem 8. Thus, we begin by checking these conditions, and then call Algorithm 2 with the models restricted to the shared alphabet. For example, consider models $\mathbb{Z}$ and A in Figure 6.3 and assume that $\alpha\mathbb{Z} = \{\mathtt{a}, \mathtt{c}\}$ and

$\alpha\mathsf{A} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. These systems are inconsistent, because $\mathbb{Z}_0 \xrightarrow{a}_r \mathbb{Z}_1$ and $\mathbb{Z}_1 \overset{c}{\not\Longrightarrow}$, whereas A can still perform a transition on c after every transition on a. They also satisfy the conditions of Theorem 8, and therefore there is an $\mathcal{L}_\mu^w$ property that distinguishes $\mathbb{Z}@\{\mathsf{a}, \mathsf{c}\}$ and $\mathsf{A}@\{\mathsf{a}, \mathsf{c}\}$. Similar to the previous example, it can verified that Algorithm 2 called with parameters $\mathbb{Z}@\{\mathsf{a}, \mathsf{c}\}$ and $\mathsf{A}@\{\mathsf{a}, \mathsf{c}\}$ returns $\phi = \langle a\rangle_o\langle c\rangle_o\bot$, which is *true* in $\mathbb{Z}@\{\mathsf{a}, \mathsf{c}\}$ and *false* in $\mathsf{A}@\{\mathsf{a}, \mathsf{c}\}$.

Observe that Lines 2, 14 and 18 in Figure 6.4 must determine whether two systems are inconsistent. Instead of calling Algorithm 1 each time, it is possible to modify Algorithm 1 to compute a list of reachable inconsistent pairs of MTSs (i.e., an “inconsistency relation”) with respect to the original systems, which makes the consistency check in Algorithm 2 an $O(1)$ operation. For example, $\{(\mathbb{Z}_0, \mathsf{A}_0)$,$(\mathbb{Z}_1, \mathsf{A}_1)$,$(\mathbb{Z}_1, \mathsf{A}_2))\}$ is such a list for $\mathbb{Z}$ and A in the previous example. Furthermore, each iteration of the For loop on Line 6 is $O(|T|^2)$, and the worst-case complexity occurs when all pairs are `visited`, making the algorithm $O(|S_M| \times |S_N| \times |T|^2)$.

6.4 Building a Common Refinement

In this section, we introduce the $+_{cr}$ operator and show that if M and N are consistent, $M +_{cr} N$ builds an element of $CR(M, N)$, which preserves the properties of the original systems. Although $+_{cr}$ does not build an MCR in general, we give sufficient conditions for it to build the LCR, if it exists.

Definition 21. (The $+_{cr}$ operator) *Let* $M = (S_M, L_M, \Delta^r_M, \Delta^p_M, s_{0M})$ *and* $N = (S_N, L_N, \Delta^r_N, \Delta^p_N, s_{0N})$ *be MTSs and let* C_{MN} *be the largest consistency relation between them.* $M +_{cr} N$ *is the MTS* $(S_M \times S_N, L_M \cup L_N \cup \{\epsilon\}, \Delta^r, \Delta^p, (s_{0M}, s_{0N}))$*, where* Δ^r *and* Δ^p *are the smallest relations that satisfy the rules given in Figure 6.5, assuming that* $\{(M, N)$,$(M', N') \subseteq C_{MN}$.

Remark 5. *In Definition 21, rules TM, MT, TT, and MM may introduce self-loops on* τ*, but we assume they are subsequently removed because they do not affect observable behaviour.*

$$\text{TD}\ \frac{M \overset{\ell}{\Longrightarrow}_r M',\ N \overset{\epsilon}{\Longrightarrow}_p N'}{M +_{cr} N \xrightarrow{\ell}_r M' +_{cr} N'}\ \ell \notin \alpha N \cup \{\tau\} \qquad \text{DT}\ \frac{M \overset{\epsilon}{\Longrightarrow}_p M',\ N \overset{\ell}{\Longrightarrow}_r N'}{M +_{cr} N \xrightarrow{\ell}_r M' +_{cr} N'}\ \ell \notin \alpha M \cup \{\tau\}$$

$$\text{TM}\ \frac{M \overset{\hat{\ell}}{\Longrightarrow}_r M',\ N \overset{\hat{\ell}}{\Longrightarrow}_m N'}{M +_{cr} N \xrightarrow{\ell}_r M' +_{cr} N'}\ \ell \in Act_\tau \qquad \text{MT}\ \frac{M \overset{\hat{\ell}}{\Longrightarrow}_m M',\ N \overset{\hat{\ell}}{\Longrightarrow}_r N'}{M +_{cr} N \xrightarrow{\ell}_r M' +_{cr} N'}\ \ell \in Act_\tau$$

$$\text{MD}\ \frac{M \overset{\ell}{\Longrightarrow}_m M',\ N \overset{\epsilon}{\Longrightarrow}_p N'}{M +_{cr} N \xrightarrow{\ell}_r M' +_{cr} N'}\ \ell \notin \alpha N \cup \{\tau\} \qquad \text{DM}\ \frac{M \overset{\epsilon}{\Longrightarrow}_p M',\ N \overset{\ell}{\Longrightarrow}_m N'}{M +_{cr} N \xrightarrow{\ell}_r M' +_{cr} N'}\ \ell \notin \alpha M \cup \{\tau\}$$

$$\text{TT}\ \frac{M \overset{\hat{\ell}}{\Longrightarrow}_r M',\ N\ \overset{\hat{\ell}}{\Longrightarrow}_r N'}{M +_{cr} N \xrightarrow{\ell}_r M' +_{cr} N'}\ \ell \in Act_\tau \qquad \text{MM}\ \frac{M \overset{\hat{\ell}}{\Longrightarrow}_m M', N \overset{\hat{\ell}}{\Longrightarrow}_m N'}{M +_{cr} N \xrightarrow{\ell}_m M' +_{cr} N'}\ \ell \in Act_\tau$$

Figure 6.5: Rules for the $+_{cr}$ operator.

Intuitively, the areas of agreement (described by the consistency relation) of the models being merged are run in parallel, synchronizing on shared actions and producing transitions in the merged model that amount to merging knowledge from both models. Thus, maybe transitions in one model can be overridden by transitions that are required or proscribed in the other. For example, if M can transit on ℓ through a required transition and N can do so via a maybe transition, then $M +_{cr} N$ can transit on ℓ through a required transition, captured by rules TM and MT in Figure 6.5. Also, if M can transit on a shared action ℓ through a required transition and N cannot transit on ℓ, then $M +_{cr} N$ cannot either.

The cases in which the models agree on a transition are handled by rules TT and MM in Figure 6.5. If both M and N can transit on ℓ through required transitions, then $M +_{cr} N$ can transit on ℓ through a required transition, and similarly for maybe transitions.

The rules discussed so far construct a merge that is a CR of the original models. Required transitions are preserved in the composition, and possible transitions are introduced only if one of the original models has a possible transition. We now address states in which the models disagree on whether the action is required or proscribed, i.e., for some $a \in (\alpha M \cap \alpha N) \cup \{\tau\}$, either (1) $M \xrightarrow{a}_r$ and $N \not\xrightarrow{a}$ or (2) $M \not\xrightarrow{a}$ and $N \xrightarrow{a}_r$. Suppose that $M \xrightarrow{a}_r$ and $N \not\xrightarrow{a}$, yet M and N are consistent. If we allow $M +_{cr} N$ to transit on a, i.e., $M +_{cr} N \xrightarrow{a}_p$, then $(M +_{cr} N)@\alpha N$ is not a refinement of N. However, we must ensure that

$(M +_{cr} N)@\alpha M$ is a refinement of M. Since $M \xrightarrow{a}_{r}$, by Condition (1) in Definition 17, we are guaranteed that $N@\alpha M \xRightarrow{\hat{a}}_{p}$, and therefore $(M +_{cr} N)@\alpha M$ should be able to transit through required transitions on τ to a state in which it can perform a required transition on a: $(M +_{cr} N)@\alpha M \xRightarrow{\hat{a}}_{r}$.

The rules mentioned so far do not apply to non-shared actions. If $\ell \neq \tau$ is not in a model alphabet, then that model is not concerned with ℓ. Therefore, if one model can transit on a non-shared action ℓ through a required transition, the merge can as well, and the other model may move internally on zero or more τ transitions. This is captured by rules TD and DT in Figure 6.5. For example, consider models D and E in Figure 6.3 and assume that $\alpha\mathsf{D} = \{\mathsf{a}, \mathsf{b}\}$ and $\alpha\mathsf{E} = \{\mathsf{a}\}$. The consistency relation computed by Algorithm 1 in Figure 6.2 is $C_{\mathsf{DE}} = \{(\mathsf{D}_0, \mathsf{E}_0), (\mathsf{D}_1, \mathsf{E}_1)\}$. $\mathsf{D} \xrightarrow{a}_{r} \mathsf{D}_1$, but $(\mathsf{D}_1, \mathsf{E}_0) \notin C_{\mathsf{DE}}$, and therefore $\mathsf{D}_0 +_{cr} \mathsf{E}_0 \not\xrightarrow{a} \mathsf{D}_1 +_{cr} \mathsf{E}_0$. However, $\mathsf{E}_0 \xrightarrow{\tau}_{r} \mathsf{E}_1$, and $(\mathsf{D}_1, \mathsf{E}_1) \in C_{\mathsf{DE}}$. By rule TD, $\mathsf{D} +_{cr} \mathsf{E}$ has a required transition on a, i.e., $\mathsf{D}_0 +_{cr} \mathsf{E}_0 \xrightarrow{a}_{r} \mathsf{D}_1 +_{cr} \mathsf{E}_1$. In fact, $\mathsf{D} +_{cr} \mathsf{E}$ is precisely D, which is in $\mathcal{CR}(\mathsf{D},\mathsf{E})$.

Similar reasoning explains rules MD and DM, except if one model can transit on ℓ through a *maybe* transition, then the merge constructed with $+_{cr}$ can transit on a *required* transition. That is, $+_{cr}$ is conservative with respect to rules DM and MD.

Special care must be taken in order to combine only consistent behaviours of the two systems (i.e., elements in the consistency relation). For example, suppose that $\mathsf{A} +_{cr} \mathsf{A}$ (see Figure 6.3) is built without this restriction. There are two transitions on a from the initial state of A, and therefore four ways of combining them with the rules in Figure 6.5. This composition results in model C, which is *not* a refinement of A. On the other hand, the pairs $(\mathsf{A}_1, \mathsf{A}_2)$ and $(\mathsf{A}_2, \mathsf{A}_1)$ are not in any consistency relation between A and itself, and so abiding by the restriction, $\mathsf{A} +_{cr} \mathsf{A}$ is equivalent to A, as desired.

The operator $+_{cr}$ builds a CR of any two consistent models.

Theorem 11. ($+_{cr}$ builds common refinements) *If M and N are consistent, then $M +_{cr} N$ is in $\mathcal{CR}(M, N)$.*

Proof:

Assume M and N are consistent and let C_{MN} be the largest consistency relation between them. We show that $M \preceq_o (M +_{cr} N)@\alpha M$ and $N \preceq_o (M +_{cr} N)@\alpha N$.

$M \xrightarrow{\ell}_r M' \land \ell \in (\alpha M \cap \alpha N) \cup \{\tau\} \land (M, N) \in C_{MN}$

(Condition (1) in Definition 17)

$\Rightarrow \quad \exists x_1, x_2, \ldots, x_i, y_1, y_2, \ldots, y_j \in (\mathit{Act}_\tau \setminus \alpha M) \cdot$

$N \xRightarrow{\hat{x_1}}_p N_{x_1} \cdots \xRightarrow{\hat{x_i}}_p N_{x_i} \xRightarrow{\hat{\ell}}_p N_{y_1} \xRightarrow{\hat{y_1}}_p N_{y_2} \cdots \xRightarrow{\hat{y_j}}_p N'$

$\land \forall k \cdot (M, N_{x_k}) \in C_{MN} \land \forall k \cdot (M', N_{y_k}) \in C_{MN} \land (M', N') \in C_{MN}$

(Rules DM, DT, TM, TT in Figure 6.5 and $\{(M, N_{x_k}), (M', N_{y_k}), (M', N')\} \subseteq C_{MN}$)

$\Rightarrow \quad (M +_{cr} N) \xrightarrow{x_1}_r \cdots \xrightarrow{x_i}_r (M +_{cr} N_{x_i}) \xrightarrow{\ell}_r (M' +_{cr} N_{y_1})$

$\xrightarrow{y_1}_r \cdots \xrightarrow{y_j}_r (M' +_{cr} N') \land (M', N') \in C_{MN}$

(Definition 4)

$\Rightarrow \quad (M +_{cr} N)@\alpha M \xRightarrow{\hat{\ell}}_r (M' +_{cr} N')@\alpha M \land (M, N') \in C_{MN}.$

$M \xrightarrow{\ell}_r M' \land \ell \in (\alpha M \setminus \alpha N) \land (M, N) \in C_{MN}$

(Condition (3) in Definition 17)

$\Rightarrow \quad \exists N' \cdot N \xRightarrow{\epsilon}_p N' \land (M', N') \in C_{MN}$

(Rule TD in Figure 6.5 and $(M, N') \in C_{MN}$)

$\Rightarrow \quad (M +_{cr} N) \xrightarrow{\ell}_r (M' +_{cr} N') \land (M', N') \in C_{MN}$

(Definition 4)

$\Rightarrow \quad (M +_{cr} N)@\alpha M \xRightarrow{\ell}_r (M' +_{cr} N')@\alpha M \land (M', N') \in C_{MN}.$

In a similar fashion, it can be shown that:

$$(M +_{cr} N)@\alpha M \xrightarrow{\ell}_p (M' +_{cr} N')@\alpha M \land (M, N) \in C_{MN} \Rightarrow M \xRightarrow{\hat{\ell}}_p M' \land (M', N') \in C_{MN}.$$

Thus, $M \preceq_o (M +_{cr} N)@\alpha M$. The proof that $N \preceq_o (M +_{cr} N)@\alpha N$ is analogous, and hence $M +_{cr} N \in \mathcal{CR}(M, N)$. □

Suppose we are interested in computing $\mathsf{A} + \mathsf{B}$, where A and B are in Figure 6.3, and $\alpha\mathsf{A} = \alpha\mathsf{B} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. The largest consistency relation is $C_{\mathsf{AB}} = \{(\mathsf{A}_0, \mathsf{B}_0), (\mathsf{A}_2, \mathsf{B}_1), (\mathsf{A}_3, \mathsf{B}_2)\}$, and A_ϵ and B_ϵ are in Figure 6.3, which we use to compute the merged model via simple transitions. Since $0 \xrightarrow{a}_m 2$ in A_ϵ, $0 \xrightarrow{a}_r 1$ in B_ϵ, and $(\mathsf{A}_2, \mathsf{B}_1) \in C_{\mathsf{AB}}$, it follows that $(0, 0) \xrightarrow{a}_r (2, 1)$ in $\mathsf{A} +_{cr} \mathsf{B}$ by rule MT. Since $2 \xrightarrow{c}_r 3$ in A_ϵ, $1 \xrightarrow{c}_r 2$ in B_ϵ, and $(\mathsf{A}_3, \mathsf{B}_2) \in C_{\mathsf{AB}}$, it follows that $(2, 1) \xrightarrow{c}_r (3, 2)$ in $\mathsf{A} +_{cr} \mathsf{B}$ by rule TT. Additionally, the self-loops on ϵ in both models are matched by rule TT, but are subsequently removed (see Remark 5). Hence, $\mathsf{A} +_{cr} \mathsf{B} = \mathsf{B}$, as desired.

Refer to the models shown in Figure 3.2 of Chapter 3. $\mathcal{I} +_{cr} \mathcal{J} = \mathcal{O}$, but as discussed in Chapter 3, MCRs of $\mathcal{I}$ and $\mathcal{J}$ are $\mathcal{K}$ and $\mathcal{L}$. Therefore, the $+_{cr}$ operator is imprecise: it computes a CR that is not necessarily minimal, because rules DM and MD convert all non-shared maybe transitions to required transitions in the composition. We address this problem by introducing another operator in Section 6.5. On the other hand, if the LCR exists, then $+_{cr}$ builds it when the determinacy condition holds (Definition 19).

Theorem 12. (Sufficient Conditions for $+_{cr}$ to build the LCR) *If $\mathcal{LCR}_{M,N}$ exists and M and N satisfy the determinacy condition, then $M +_{cr} N$ is $\mathcal{LCR}_{M,N}$.*

Proof:

If $\mathcal{LCR}_{M,N}$ exists, the non-shared condition (Definition 20) and the multiple-maybe condition (Definition 18) are satisfied: no choice with respect to the order of non-shared actions can be made, and no non-shared maybe transitions must be converted to required transitions. Since the determinacy condition holds, all choices with respect to non-determinism are equivalent, and therefore $+_{cr}$ builds the LCR. □

To see why the determinacy condition is in Theorem 12, consider model H in Figure 6.3. Clearly, $\mathcal{LCR}_{\mathsf{H},\mathsf{H}}$ exists, but $(\mathsf{H}_0 +_{cr} \mathsf{H}_0) \xrightarrow{a}_r (\mathsf{H}_1 +_{cr} \mathsf{H}_0) \xrightarrow{b}_r (\mathsf{H}_1 + \mathsf{H}_0)$, whereas H cannot transit on a required $\mathtt{a}$ followed by a required $\mathtt{b}$, so $\mathsf{H} +_{cr} \mathsf{H} \not\equiv_o \mathsf{H}$.

Finally, assuming the largest consistency relation and the observation graphs have been computed, the number of transitions in $M +_{cr} N$ is $O(|T_M| \times |T_N|)$, where $|T_i| = |L_i| \times |S_i|^2$,

$$\text{MD}\,\frac{M \overset{\ell}{\Longrightarrow}_m M',\ N \overset{\epsilon}{\Longrightarrow}_p N'}{M +_{ca} N \xrightarrow{\ell}_m M' +_{ca} N'}\ \ell \notin \alpha N \cup \{\tau\} \qquad \text{DM}\,\frac{M \overset{\epsilon}{\Longrightarrow}_p M',\ N \overset{\ell}{\Longrightarrow}_m N'}{M +_{ca} N \xrightarrow{\ell}_m M' +_{ca} N'}\ \ell \notin \alpha M \cup \{\tau\}$$

Figure 6.6: Rules for the $+_{ca}$ operator.

and therefore the rules in Figure 6.5 are applied at most $O(|T_M| \times |T_N|))$ times. Furthermore, removing subsequent self-loops on τ is an $O(|T_M| \times |T_N|)$ operation, and thus computing $M +_{cr} N$ is $O(|T_M| \times |T_N|)$. The operator $+_{ca}$ defined Section 6.5 also shares this complexity.

6.5 Building a Common Abstraction of Minimal Common Refinements

The operator $+_{cr}$ introduces imprecision through rules DM and MD by converting all non-shared maybe transitions into required transitions in the composition. Consequently, it does not build a minimal common refinement in all cases. The operator $+_{ca}$, defined below, takes the opposite approach of leaving all non-shared maybe transitions as maybe in the composition.

Definition 22. (The $+_{ca}$ operator) *The $+_{ca}$ operator is defined in same way as $+_{cr}$ in Definition 21, except rules MD and DM of Figure 6.5 are replaced by those in Figure 6.6.*

The operator $+_{ca}$ does not build CRs. For example, refer to models F and G in Figure 6.3 and assume that $\alpha\mathsf{F} = \{\mathsf{c}\}$ and $\alpha\mathsf{G} = \{\mathsf{a}, \mathsf{c}\}$. $\mathsf{F} +_{ca} \mathsf{G}$ is G, and $\mathsf{G}@\{\mathsf{c}\}$ is not a refinement of F because $\mathsf{F} \xrightarrow{c}_r$ and $\mathsf{G}@\{\mathsf{c}\} \overset{c}{\Longrightarrow}_m$. Instead, it is reasonable to expect that $+_{ca}$ builds a common abstraction of all MCRs (see Definition 16 in Chapter 3) of the models being merged, due to rules DM and MD in Figure 6.6. However, it may not be the case in general. For example, recall from Section 4.2 that $\mathbb{D}$ is an MCR of models $\mathbb{A}$ and $\mathbb{B}$ (see Figure 4.1). By rules TD and DT and the fact that $\mathbb{A}_0$ and $\mathbb{B}_1$ are consistent, $\mathbb{A}_0 +_{ca} \mathbb{B}_0 \xrightarrow{b}_r \mathbb{A}_0 +_{ca} \mathbb{B}_1$. Since $\mathbb{D} \overset{b}{\not\Longrightarrow}$, model $\mathbb{D}$ does not refine $\mathbb{A} +_{ca} \mathbb{B}$, and therefore $\mathbb{A} +_{ca} \mathbb{B}$ is not a CA of all MCRs of $\mathbb{A}$ and $\mathbb{B}$. Intuitively, to find an MCR of $\mathbb{A}$ and $\mathbb{B}$, a choice must be made with respect to the order of the transition on c in $\mathbb{A}$ and the transition on b in $\mathbb{B}$. Model $\mathbb{D}$ corresponds to putting the transition on c first, but the operator $+_{ca}$ always makes both choices, resulting in model $\mathbb{E}$ that is not a CA of all MCRs. Hence, we must limit non-shared behaviour of the models being merged in order to restrict the existence of such choices, i.e., we must assume that the non-shared condition (Definition 20) holds. Similarly, we also require that the determinacy condition (Definition 19) holds (e.g., for the models in Figure 3.2, $\mathcal{W} +_{ca} \mathcal{X} \not\preceq_o \mathcal{Y}$, where $\mathcal{Y} \in \mathcal{MCR}(\mathcal{W}, \mathcal{X})$).

Theorem 13. ($+_{ca}$ is a common abstraction of all MCRs) *If M and N are consistent MTSs that satisfy the determinacy condition and the non-shared condition, then:*

$$\forall Q \in \mathcal{MCR}(M, N) \cdot M +_{ca} N \preceq_o Q@(M +_{ca} N).$$

Proof:

Let $Q \in \mathcal{MCR}(M, N)$. The given conditions guarantee that Q and $M +_{ca} N$ differ (up to observational equivalence) only in the application of rules DM and MD: some non-shared maybe transitions in $M +_{ca} N$ may be required transitions in Q. □

Hence, $M +_{ca} N$ approximates $\mathcal{GCA}_{M,N}$ from below: $M +_{ca} N \preceq_o \mathcal{GCA}_{M,N}$. Unfortunately, $+_{ca}$ does not compute the actual GCA. For example, consider models B, F, and G in Figure 6.3 with $\alpha\mathsf{F} = \{\mathsf{c}\}$ and $\alpha\mathsf{B} = \alpha\mathsf{G} = \{\mathsf{a}, \mathsf{c}\}$. $\mathcal{LCR}_{\mathsf{F},\mathsf{G}}$ is B, and therefore, by Remark 2, $\mathcal{GCA}_{\mathsf{F},\mathsf{G}}$ is also B. Since $\mathsf{F} +_{ca} \mathsf{G} = \mathsf{G}$, $+_{ca}$ produces a model that is refined by, but is not equivalent to the GCA. On the other hand, if rules DM and MD are never applied, then $M +_{ca} N = M +_{cr} N$. Therefore, by Theorem 12 and Remark 2, if the LCR exists, M and N satisfy the determinacy condition, and rules DM and MD are not applied, then $+_{ca}$ builds the LCR, which is equivalent to the GCA.

In summary, it is not always clear which non-maybe transitions should be converted to required and which should be left as maybe. If all are converted, as is the case for $+_{cr}$, then minimality is lost. In fact, the correct rules for computing the GCA of all MCRs are somewhere between rules MD and DM of $+_{cr}$ and $+_{ca}$ (i.e., some should be converted to required and some should stay maybe in the composition). Preliminary work on this problem has been addressed in [49] in the form of an algorithm that supports the elaboration of a CA into a desired MCR. In particular, the algorithm helps the modeller choose which non-shared maybe transitions to convert to required transitions in the merge. We leave for future work exploring this problem in more detail.

6.6 Discussion

Throughout this chapter, we have developed algorithms to support the merge process in Figure 6.1. Below, we summarize our results with respect to this process, and discuss how they relate to automating merge.

Given MTSs M and N, consistency can be decided by Algorithm 1 (Line 1). If M and N are inconsistent, after checking the sufficient conditions in Theorem 8, Algorithm 2 can be used to return a property that distinguishes between them (Line 8). If the conditions fail, then Algorithm 1 can be easily modified to return the inconsistent states as an alternative form of feedback. If M and N are consistent, Algorithm 1 builds the largest consistency relation between them, which, in some cases, may be modified due to unwanted matches. If minimality is not required (Line 2), $+_{cr}$ constructs a CR that preserves the desired properties and can be returned (Line 3). Otherwise, a minimal merge is required. If $\mathcal{LCR}_{M,N}$ exists (which can decided in most cases using Theorem 3) and M and N satisfy the determinacy condition, $M +_{cr} N$ is their LCR and can be returned (Line 5). On the other hand, if $\mathcal{LCR}_{M,N}$ does not exist and M and N satisfy the non-shared condition and the determinacy condition, then a CA to all MCRs of M and N can be constructed with $+_{ca}$ (Line 6). Note that $+_{ca}$ does not construct the GCA, as desired, but can still be elaborated into an MCR, or used to construct the set of MCRs [49] (Line 6). We leave for future work improving $+_{ca}$ to construct the GCA.

In the above process, some assumptions on the MTSs must hold in order to ensure complete automation, e.g., the non-shared condition and determinacy condition must hold for $+_{ca}$ to produce a CA. However, theses cases naturally correspond to the cases that require human intervention. For example, we have noted that $+_{ca}$ cannot make decisions with respect to the order of non-shared actions, and therefore these decisions must be made by appropriately modifying the consistency relation. The only case that requires an assumption that is not a consequence of human interaction is that the determinacy condition must hold on Line 5, which we plan on weakening in future work. Hence, our algorithms fully support the process outlined in Figure 6.1, employing automation when possible.

Chapter 7

A Case Study: the Mine Pump

In this chapter, we apply our results to a case study known as the mine pump [34]. In Section 7.1, we give a high level overview and introduce some components of the case study. In Section 7.2, we show that by using our algorithms for detecting inconsistencies between stakeholders' models, we can gain insight into whether the desired properties of the stakeholders are in fact reasonable. Finally, Section 7.3 shows that by relaxing the required properties from the stakeholders (based on the insight gained in Section 7.2), we can merge updated consistent models from the stakeholders to create an overall model that satisfies the desired behaviours.

7.1 Description

A pump controller is used to prevent the water in a mine sump from passing some threshold, and hence flooding the mine. To avoid the risk of explosion, the pump may only be active when there is no methane gas present in the mine. The pump controller monitors the water and methane levels by communicating with two sensors.

We model the mine pump with four components: *WaterLevelSensor*, *MethaneSensor*, *PumpControl*, and *Pump*. The complete system, *MinePump*, is the parallel composition of these components. *WaterLevelSensor* models the water sensor and includes assumptions on how the water level is expected to change between low, medium, and high. *Methane* keeps track

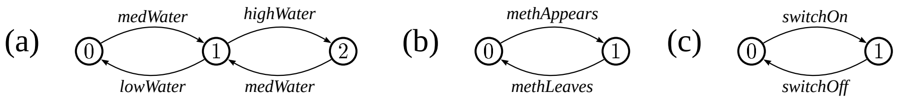

Figure 7.1: The MTSs for: (a) *WaterLevelSensor*, (b) *MethaneSensor*, and (c) *Pump*.

of whether methane is present in the mine, and *Pump* models the physical pump that can be switched on and off. The LTSs for these components are shown in Figure 7.1, where we assume initially that the water is low, the pump is off, and no methane is present.

PumpControl describes the controller that monitors water and methane levels, and controls the pump in order to guarantee the safety properties of the pump system. Note that the informal description given above leaves open the exact water level at which to turn the pump on and off. For example, the pump could be turned on when there is high water or possibly when the water is not low, (e.g., at a medium level). The pump could be turned off when there is low water or possibly when the water is not high. In what follows, we investigate models for the pump controller from different stakeholders, which are intended to be merged to create a model of the entire system, namely *MinePump*.

7.2 On and Off Policies: A First Attempt

In this section, we assume that there are two stakeholders for the pump controller: one with the knowledge of when the pump should be on (referred to as the on policy) and another with knowledge of when the pump should be off (referred to as the off policy). The stakeholders' properties are shown in Table 7.1, and are expressed in FLTL using the following fluents:

$$\begin{aligned}
\mathit{AtHighWater} &= \langle \mathit{highWater}, \{\mathit{lowWater}, \mathit{medWater}\}\rangle,\\
\mathit{AtLowWater} &= \langle \mathit{lowWater}, \{\mathit{medWater}, \mathit{highWater}\}\rangle \ \mathit{Initially\ true},\\
\mathit{PumpOn} &= \langle \mathit{switchOn}, \mathit{switchOff}\rangle,\\
\mathit{MethanePresent} &= \langle \mathit{methAppears}, \mathit{methLeaves}\rangle.\{\mathit{switchOn}\}\rangle,
\end{aligned}$$

Φ_1 :	$\mathbf{G}(\mathit{PumpOn} \Rightarrow (\neg \mathit{switchOn}\ \mathbf{U}\ \neg \mathit{PumpOn}))$ (the pump can only be turned on if it is currently off)
Φ_2 :	$\mathbf{G}(\neg \mathit{PumpOn} \Rightarrow (\neg \mathit{switchOff}\ \mathbf{U}\ \mathit{PumpOn}))$ (the pump can only be turned off if it is currently on)
Φ_3 :	$\mathbf{G}(\mathit{AtHighWater} \wedge \neg \mathit{MethanePresent} \Rightarrow \mathbf{X}\ \mathit{PumpOn})$ (the on policy: the pump is turned on when there is high water and no methane)
Φ_4 :	$\mathbf{G}(\mathit{AtLowWater} \vee \mathit{MethanePresent} \Rightarrow \mathbf{X} \neg \mathit{PumpOn})$ (the off policy: the pump is turned off when there is low water or methane present)

Table 7.1: Desired properties of *MinePump*.

Both stakeholders wish to satisfy properties Φ_1 and Φ_2, which express that the pump should only be turned on if it is already off, and should only be turned off if it is already on, respectively. In addition, the stakeholder for the on policy desires to satisfy property Φ_3, which expresses that when there is high water and no methane, the pump should be immediately turned on if it is not already. The stakeholder for the off policy desires to satisfy property Φ_4, which expresses that if there is low water or methane present, the pump should be immediately turned off if it is not already.

The stakeholders' models are *OnPolicy* and *OffPolicy*, shown in Figure 7.2, which use the abbreviation *WATER* = $\{\mathit{lowWater}, \mathit{medWater}, \mathit{highWater}\}$. *OnPolicy* turns the pump on when there is high water and no methane present, and leaves the possibility open for turning the pump on when there is medium water. *OffPolicy* turns the pump off when there is low water or methane appears. In addition, *OffPolicy* models a danger light with actions *dangerLightOn* and *dangerLightOff* (unobservable to *OnPolicy*), which is turned on when methane is present in the mine.

Both *OnPolicy* and *OffPolicy* can be composed with *PumpControl*, *MethaneSensor*, and *WaterLevelSensor* to obtain two partial models of the entire system, referred to as $\mathit{MinePump}_1$ and $\mathit{MinePump}_2$ (see Appendix A). Properties Φ_1 to Φ_3 are *true* in $\mathit{MinePump}_1$, and Φ_1, Φ_2, and Φ_4 are *true* in $\mathit{MinePump}_2$, as desired. Note that to specify the *OnPolicy*, it is necessary to refer to the event *switchOff* in order to ensure that Φ_1 and Φ_2 are satisfied. However, *OnPolicy* leaves

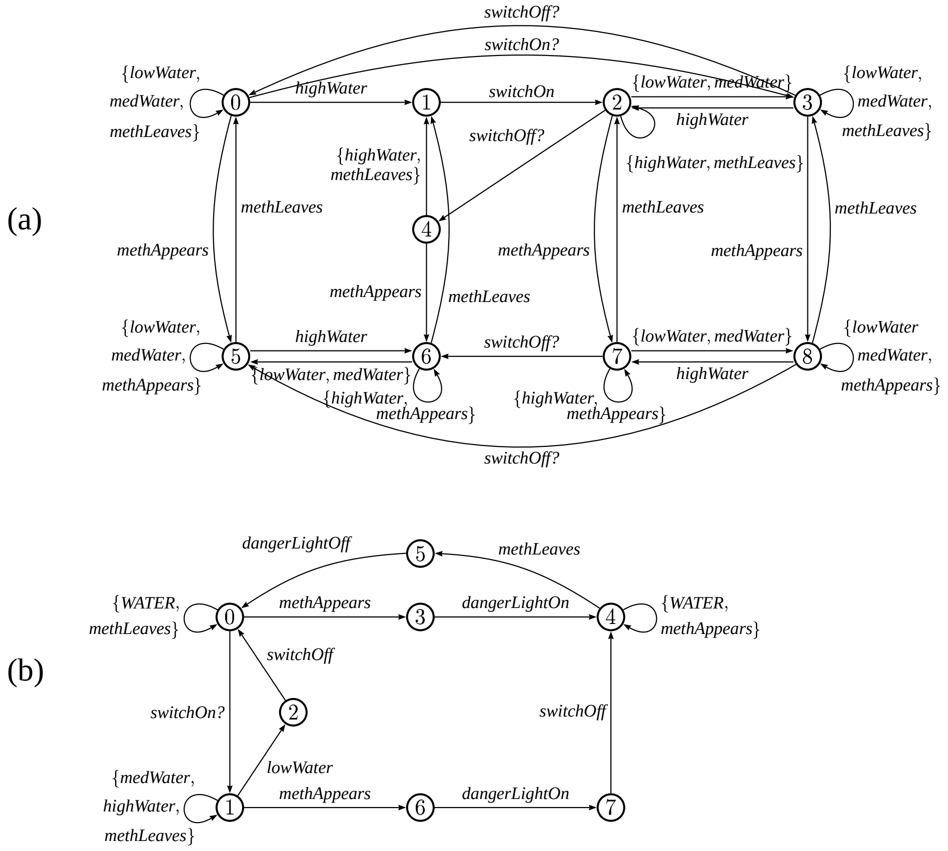

Figure 7.2: The MTSs for: (a) *OnPolicy*, and (b) *OffPolicy*.

the off policy open by modelling possibilities for turning the pump off with maybe behaviour, and similarly, *OffPolicy* leaves the on policy open. Hence, properties Φ_3 and Φ_4 evaluate to *maybe* in *MinePump*$_2$ and *MinePump*$_1$, respectively.

Our goal is to build *MinePump* using the knowledge contained in *OnPolicy* and *OffPolicy*, while preserving the properties of the individual stakeholders' models *MinePump*$_1$ and *MinePump*$_2$. In other words, we wish to construct *MinePump* $=$ (*OnPolicy* $+$ *OffPolicy*) $\|$*WaterLevelSensor*$\|$ *MethaneSensor* $\|$ *Pump* and ensure that it satisfies properties Φ_1 – Φ_4.

Unfortunately, by Algorithm 1 in Section 6.2, *OnPolicy* and *OffPolicy* are inconsistent, and thus cannot be merged. To understand the sources of inconsistencies, we note that these models satisfy Conditions (1) and (2) in Theorem 8. Therefore, there exists an $\mathcal{L}_\mu^w$ property that distinguishes *OnPolicy* and *OffPolicy* when restricted to the shared alphabet $X = \{$*lowWater*, *medWater*, *highWater*, *methLeaves*, *methAppears*, *switchOn*, *switchOff*$\}$, and we can use Algorithm 2 in Section 6.3 to find one. In particular, one such property is:

$$\langle \mathit{highWater}\rangle_o\langle \mathit{switchOn}\rangle_o\langle \mathit{methAppears}\rangle_o\langle \mathit{methAppears}\rangle_o\mathbf{t}.$$

This property is *true* in *OnPolicy@*X since $0 \xrightarrow{\mathit{highWater}}_r 1 \xrightarrow{\mathit{switchOn}}_r 2 \xrightarrow{\mathit{methAppears}}_r 7 \xrightarrow{\mathit{methAppears}}_r 7$, and *false* in *OffPolicy@*X since after following the observable trace *highWater*, *switchOn*, and *methAppears*, either state 6 or 7 is reached, both of which proscribe the action *methAppears*. This property highlights the fact that in *OffPolicy*, whenever the pump is on and methane appears, the danger light is *immediately* turned on and the pump is *immediately* switched off: states 6 and 7 in *OffPolicy* proscribe all actions other than *dangerLightOn* and *switchOff*. Another distinguishing property is:

$$\langle \mathit{methLeaves}\rangle_o\langle \mathit{highWater}\rangle_o\langle \mathit{medWater}\rangle_o\mathbf{f},$$

which expresses the fact that when methane is absent and the water is high, it may no longer be possible for the water to move to a medium level. This property is *true* in *OnPolicy@*X since $0 \xrightarrow{\mathit{methLeaves}}_r 0 \xrightarrow{\mathit{highWater}}_r 1$ and $1 \overset{\mathit{medWater}}{\not\Longrightarrow}$, and *false* in *OffPolicy@*X since every transition on *methLeaves* and *highWater* from state 0 is a self-loop, and $0 \xrightarrow{\mathit{medWater}}_r$. Upon inspection, we see that this inconsistency is due to the fact that *OnPolicy immediately* turns the pump on whenever there is high water and no methane is present.

One possible resolution to the above inconsistencies is to model some of the the self-loops in *OnPolicy* and *OffPolicy* with maybe behaviour (e.g., in state 7 of *OnPolicy* and state 0 of *OffPolicy*), but this unnecessarily reduces the definite behaviour of both models and does not address the source of the inconsistencies: properties Φ_3 and Φ_4 in Table 7.1 are too strong, because they utilize the next operator, which is not closed under stuttering [1]. In particular, this forces the presence of immediacy in the on and off policies, which was shown to ultimately cause inconsistencies. We discuss this case in Section 7.3, where we weaken properties Φ_3 and Φ_4 to obtain consistent pump control policies.

$\Phi'_3 : \mathbf{G}((\mathit{AtHighWater} \wedge \neg \mathit{MethanePresent})\ \mathbf{U}\ \mathit{tick} \Rightarrow \neg \mathit{tick}\ \mathbf{U}\ \mathit{PumpOn})$
(the on policy: if there is high water and no methane for a given amount of time, the pump is on)
$\Phi'_4 : \mathbf{G}((\mathit{AtLowWater} \vee \mathit{MethanePresent})\ \mathbf{U}\ \mathit{tick} \Rightarrow \neg \mathit{tick}\ \mathbf{U} \neg \mathit{PumpOn})$
(the off policy: if there is low water or methane for a given amount of time, the pump is off)

Table 7.2: Desired Properties of *MinePumpNoIm*.

7.3 On and Off Policies: A Second Attempt

In the previous section, we saw that explicit immediacy caused inconsistencies in the pump controller models. We therefore desire a weaker form of immediacy: one that still enforces the pump control policies, but does not require, for example, that the pump is turned on as soon as the proper conditions are true. Rather, if the conditions for turning the pump on hold, then other actions may occur, but before a certain amount of time passes, the pump must be turned on if the conditions for turning the pump on still hold. Hence, we must explicitly model time events in the pump controller models.

In particular, we weaken properties Φ_3 and Φ_4 in Table 7.1 to properties Φ'_3 and Φ'_4 in Table 7.2. These properties involve the special *tick* event of asynchronous FLTL [39], which is used to indicate the passage of time. In particular, Φ'_3 says that if it is the case that no methane is present and the water is high for a given amount of time, then the pump is turned on before the next *tick* event, and Φ'_4 says that when methane is present or the water is low for a given amount of time, the pump is turned off before the next *tick* event. That is, these properties do not enforce immediacy in the sense of the previous section, but rather if the respective conditions for the on and off policies hold until the next unit of time has occurred, then the status of the pump is changed before this.

As in the previous section, there are two stakeholders for the pump controller, which are composed with the components in Figure 7.1 to obtain two partial models of the entire system. The on and off policies remain the same, except that immediacy is no longer enforced and the special *tick* event is used. *OnPolicyNoIm* models the on policy, *OffPolicyNoIm* models

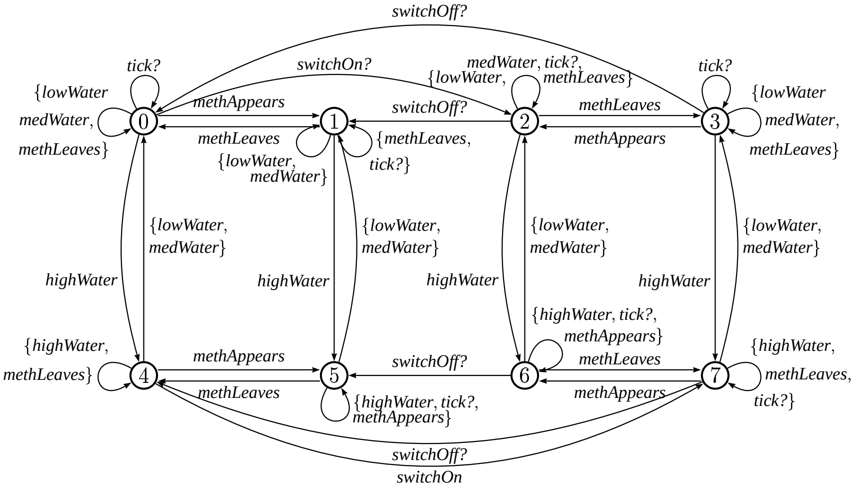

Figure 7.3: The MTS for *OnPolicyNoIm*.

the off policy (including the danger light), and *MinePumpNoIm*$_1$ and *MinePumpNoIm*$_2$ (see Appendix A) are the stakeholders' respective partial models of the entire system. The MTS for *OnPolicyNoIm* is shown in Figure 7.3, and the FSP code [41] for *OffPolicyNoIm* is given in Figure 7.4. Both *MinePumpNoIm*$_1$ and *MinePumpNoIm*$_2$ satisfy properties Φ_1 and Φ_2 in Table 7.1, and additionally *MinePumpNoIm*$_1$ satisfies Φ'_3 and *MinePumpNoIm*$_2$ satisfies Φ'_4. Similar to the previous section, Φ'_3 evaluates to *maybe* in *MinePumpNoIm*$_2$ and Φ'_4 evaluates to *maybe* in *MinePumpNoIm*$_1$.

Recall that we wish to build *MinePumpNoIm* using the information in *OnPolicyNoIm* and *OffPolicyNoIm*, while ensuring that *MinePumpNoIm* $=$ (*OnPolicyNoIm* $+$ *OffPolicyNoIm*) $\|$ *WaterLevelSensor* $\|$ *MethaneSensor* $\|$ *Pump* satisfies Φ_1, Φ_2, Φ'_3, and Φ'_4. By Algorithm 1 in Section 6.2, *OnPolicyNoIm* and *OffPolicyNoIm* are consistent, and hence the full pump controller can be defined as *PumpControl* = *OnPolicyNoIm* + *OffPolicyNoIm*. Additionally, the alphabet restrictions in Proposition 9 are satisfied by *MinePumpNoIm*$_1$ and *MinePumpNoIm*$_2$, and therefore there exists a merge of *MinePumpNoIm*$_1$ and *MinePumpNoIm*$_2$ that is refined by *MinePumpNoIm*. Hence, by Definition 15 and Theorem 4, properties Φ_1, Φ_2, Φ'_3, and Φ'_4 hold in *MinePumpNoIm*. In particular, the *maybe* properties Φ'_3 and Φ'_4 of *MinePumpNoIm*$_1$ and *MinePumpNoIm*$_2$ become *true* properties of *MinePump*, which corresponds to the on and off policies being refined into concrete behaviours in the merge.

```
OffPolicyNoIm = Q0,

Q0 =  ({lowWater, methLeaves, tick?} → Q0
      |switchOn? → Q1
      |methAppears → Q3
      |{highWater, medWater} → Q9),
Q1 =  (switchOff → Q0
      |{lowWater, methLeaves} → Q1
      |methAppears → Q2
      |{highWater, medWater} → Q10),
Q2 =  (methLeaves → Q1
      |{lowWater, methAppears} → Q2
      |switchOff → Q3
      |dangerLightOn → Q5
      |{highWater, medWater} → Q11),
Q3 =  (methLeaves → Q0
      |switchOn? → Q2
      |{lowWater, methAppears, tick?} → Q3
      |dangerLightOn → Q4
      |{highWater, medWater} → Q12),
Q4 =  ({lowWater, methAppears, tick?} → Q4
      |switchOn? → Q5
      |methLeaves → Q7
      |{highWater, medWater} → Q13),
Q5 =  (switchOff → Q4
      |{lowWater, methAppears} → Q5
      |methLeaves → Q6
      |{highWater, medWater} → Q14),
Q6 =  (dangerLightOff → Q1
      |methAppears → Q5
      |{lowWater, methLeaves} → Q6
      |switchOff → Q7
      |{highWater, medWater} → Q15),
Q7 =  (dangerLightOff → Q0
      |methAppears → Q4
      |switchOn? → Q6
      |{lowWater, methLeaves, tick?} → Q7
      |{highWater, medWater} → Q8),
Q8 =  (lowWater → Q7
      |{highWater, medWater, methLeaves} → Q8
      |tick? → Q8
      |dangerLightOff → Q9
      |methAppears → Q13
      |switchOn? → Q15),
Q9 =  (lowWater → Q0
      |{highWater, medWater, methLeaves} → Q9
      |tick? → Q9
      |switchOn? → Q10
      |methAppears → Q12),
Q10 = (lowWater → Q1
      |{highWater, medWater, methLeaves} → Q10
      |tick? → Q10
      |methAppears → Q11),
Q11 = (lowWater → Q2
      |methLeaves → Q10
      |switchOff → Q12
      |{highWater, medWater, methAppears} → Q11
      |dangerLightOn → Q14),
Q12 = (lowWater → Q3
      |methLeaves → Q9
      |tick? → Q12
      |switchOn? → Q11
      |{highWater, medWater, methAppears} → Q12
      |dangerLightOn → Q13),
Q13 = (lowWater → Q4
      |{highWater, medWater, methAppears} → Q13
      |tick? → Q13
      |methLeaves → Q8
      |switchOn? → Q14),
Q14 = (lowWater → Q5
      |switchOff → Q13
      |{highWater, medWater, methAppears} → Q14
      |methLeaves → Q15),
Q15 = (lowWater → Q6
      |dangerLightOff → Q10
      |methAppears → Q14
      |{highWater, medWater, methLeaves} → Q15).
```

Figure 7.4: The FSP code for *OffPolicyNoIm*.

Since there are no non-shared maybe transitions between *OnPolicyNoIm* and *OffPolicyNoIm*, the $+_{cr}$ and $+_{ca}$ operators build the same system, say $\mathit{PumpControl}_{cr} = \mathit{OnPolicyNoIm} +_{cr} \mathit{OffPolicyNoIm}$. By Theorem 12, we know that both merge operators build the LCR of these models, i.e., $\mathit{PumpControl}_{cr} \equiv_o \mathit{PumpControl}$. $\mathit{PumpControl}_{cr}$ can then be composed with the components in Figure 7.1 to obtain $\mathit{MinePumpNoIm}_{cr} = \mathit{PumpControl}_{cr} \parallel \mathit{WaterLevelSensor} \parallel \mathit{MethaneSensor} \parallel \mathit{Pump}$, which is equivalent to *MinePumpNoIm*. The FSP code for *MinePumpNoIm* (and therefore for $\mathit{MinePumpNoIm}_{cr}$) is shown in Figure 7.5.

Now consider the case when the LCR of *OnPolicyNoIm* and *OffPolicyNoIm* does not exist. By Theorem 11, $\mathit{PumpControl}_{cr}$ is a common refinement of *OnPolicyNoIm* and *OffPolicyNoIm*. By Proposition 9, the fact that $\mathit{PumpControl}_{cr}$ refines some MCR of *OnPoli-*

```
MinePumpNoIm = Q0,

 Q0 =  (tick? → Q0                Q1 =  (switchOff → Q0              Q2 =  (methLeaves → Q1
       |switchOn? → Q1                  |methAppears → Q2                  |dangerLightOn → Q3
       |methAppears → Q22               |medWater → Q21),                  |medWater → Q20
       |medWater → Q23),          Q4 =  (tick? → Q4                        |switchOff → Q22),
 Q3 =  (switchOff → Q4                  |methLeaves → Q5             Q5 =  (dangerLightOff → Q0
       |methLeaves → Q18                |medWater → Q17),                  |methAppears → Q4
       |medWater → Q19),          Q7 =  (tick? → Q7                        |tick? → Q5
 Q6 =  (lowWater → Q5                   |highWater → Q8                    |medWater → Q6
       |tick? → Q6                      |lowWater → Q18                    |switchOn? → Q18),
       |switchOn? → Q7                  |methAppears → Q19           Q8 =  (medWater → Q7
       |highWater → Q13                 |dangerLightOff → Q21),            |tick? → Q8
       |methAppears → Q17         Q10 = (methLeaves → Q9                   |dangerLightOff → Q9
       |dangerLightOff → Q23),          |switchOff → Q11                   |methAppears → Q16),
 Q9 =  (tick? → Q9                      |dangerLightOn → Q16         Q11 = (tick? → Q11
       |methAppears → Q10               |medWater → Q20),                  |dangerLightOn → Q12
       |medWater → Q21),          Q13 = (switchOn → Q8                     |methLeaves → Q14
 Q12 = (tick? → Q12                     |tick? → Q13                       |medWater → Q15),
       |methLeaves → Q13                |dangerLightOff → Q14),      Q14 = (switchOn → Q9
       |medWater → Q17),          Q16 = (methLeaves → Q8                   |tick? → Q14),
 Q15 = (highWater → Q11                 |switchOff → Q12             Q17 = (lowWater → Q4
       |tick? → Q15                     |medWater → Q19),                  |methLeaves → Q6
       |dangerLightOn → Q17       Q19 = (lowWater → Q3                     |highWater → Q12
       |lowWater → Q22                  |methLeaves → Q7                   |tick? → Q17),
       |methLeaves → Q23),              —highWater → Q16             Q20 = (lowWater → Q2
 Q18 = (dangerLightOff → Q1             |switchOff → Q17),                 |highWater → Q10
       |methAppears → Q3          Q22 = (methLeaves → Q0                   |switchOff → Q15
       |switchOff → Q5                  |dangerLightOn → Q4                |dangerLightOn → Q19
       |medWater → Q7),                 |medWater → Q15                    |methLeaves → Q21),
 Q21 = (lowWater → Q1                   |tick? → Q22),               Q23 = (lowWater → Q0
       |highWater → Q9                                                     |highWater → Q14
       |methAppears → Q20                                                  |methAppears → Q15
       |tick? → Q21),                                                      |switchOn? → Q21
                                                                           |tick? → Q23).
```

Figure 7.5: The FSP code for *MinePumpNoIm* and $\mathit{MinePumpNoIm}_{cr}$.

cyNoIm and *OffPolicyNoIm*, and Proposition 1, we know that the properties of $\mathit{MinePumpNoIm}_1$ and $\mathit{MinePumpNoIm}_2$ are preserved in $\mathit{MinePumpNoIm}_{cr}$. That is, we obtain an over-approximation of *MinePumpNoIm* that satisfies the desired properties. Alternatively, if the assumptions in Theorem 13 hold, the $+_{ca}$ operator can be used to obtain a common abstraction of all MCRs of *OnPolicyNoIm* and *OffPolicyNoIm*, which can then be elaborated into a given MCR [49].

In conclusion, we have shown that by using merge as the underlying notion, we can utilize partial models from different stakeholders and combine them to obtain an overall model that satisfies the desired properties. In general, it is useful that the properties of $\mathit{MinePumpNoIm}_1$ and $\mathit{MinePumpNoIm}_2$ are preserved rather than those of *OnPolicyNoIm* and *OffPolicyNoIm*, as the environment of the pump controller (for instance, the assumptions on the behaviour of the water level) may be necessary for proving certain properties. In fact, *OnPolicyNoIm* could be modified to be under-specified enough so as not to satisfy Φ_1 unless the assumptions modelled in *WaterLevelSensor* hold.

Chapter 8

Conclusion

In this chapter, we summarize the thesis, compare our work with related approaches, and discuss directions for future research.

8.1 Summary

The motivation for the work presented in this thesis comes from the need to support the elaboration of partial behavioural models. In particular, our work has been motivated by existing limitations of scenario-based model synthesis techniques, and hence we have focused on observable behaviour, rather than on model structure. However, our work could also be applicable in the context of composing models that cover different viewpoints [26] or aspects [6].

In this thesis, we have instantiated and studied a general merging framework for merge based on MTSs [49]. For such models, Chechik and Uchitel have argued that observational refinement is the formal underlying principle of model merging and that merging is a process that should produce a minimal common observational refinement of two consistent models [49]. We have characterized several aspects of this merge. Specifically, the merge of two consistent systems preserves 3-valued weak μ-calculus and FLTL properties, giving rise to a sound specification language for our framework. On the other hand, if two systems are inconsistent, we have seen that distinguishing properties can be used as a form of feedback: such properties give insight into areas of disagreement. Furthermore, to ensure that our framework supports component-wise merging of complex systems (common in practical applications), we have proven several positive and negative results with respect to algebraic properties, and related these results to modelling decisions. Most desired properties hold when merge is unique, but when multiple incomparable merges exist, the right choice of merge must be made in order to guarantee a particular property, i.e., human involvement may be required. Finally, we have described algorithms for supporting the merge process, and shown that such algorithms facilitate automation, when possible, and enable human intervention, when required.

The above results give rise a framework for partial model elaboration, based on merging MTSs, suitable for use in practice, that is fully supported algorithmically.

8.2 Related Work

Below, we survey related work along two directions: (1) merging, and (2) abstraction and property preservation with respect to partial models.

Merging. Larsen et. al. [38] introduce an operator with a behaviour similar to our merge (called *conjunction*), but defined only for MTSs over the same vocabulary with no τ transitions, and for which there is an *independence relation* (at which point the least common refinement exists). Their goal is to decompose a complete specification into several partial ones to enable compositional proofs. Although not studied in depth, the operator in [38] is based on strong refinement. We have shown that the existence of multiple MCRs introduces a number of subtle issues for a similar operator based on observational refinement. In addition, our sufficient condition for building a common refinement (i.e., the existence of a consistency relation) is more general than their notion of an independence relation, even for models over the same vocabulary. In particular, MTSs for which there is an independence relation also satisfy the determinacy condition, which is not a necessary condition for constructing merge in our framework. In [36], Larsen and Xinxin define a conjunction operator for Disjunctive MTSs (DMTSs), for which the least common refinement also exists, similar to the one in [38]. DMTSs are an extension of MTSs where the set of required transitions is a set of required hyper-transitions (i.e., a disjunction of multiple transitions on different actions). Larsen and Xinxin use a notion of a consistency relation, similar to ours, but defined over a single DMTS. Two DMTSs are then consistent if there is a consistency relation in the conjunction. However, the goal in [36] is to characterize equation solving in process algebra, rather than to support model elaboration. In particular, consistency is used to prove satisfiability of a given specification.

Hussain and Huth [27] also study the problem of finding a common refinement between multiple MTSs, but they focus on the complexity of the relevant model-checking procedures: consistency, satisfiability, and validity. Instead, our merge addresses the more general problem of supporting engineering activities in model elaboration (i.e., we see merging as the process of selecting the most appropriate common refinement). Finally, our models are more general than the models of Hussain and Huth in that we merge models with different vocabularies and τ transitions, but less general in that Hussian and Huth handle hybrid constraints (e.g., restricting the number of states a given proposition is evaluated in) and compute the set of all consistent pairs.

Nejati and Chechik [43] present a framework for merging 4-valued Kripke structures, where the fourth value indicates disagreement. Their merge is defined as a common strong refinement, and is therefore characterized by 3-valued μ-calculus. Like [38, 36], the least common refinement is guaranteed to exist between consistent models, but it is shown that it may not be expressible in a 3-valued model. In our framework, a unique merge is not guaranteed, but we maintain a high level of readability by expressing merge in a 3-valued model, making it suitable for use in practice. Nejati and Chechik also allow the models being merged to have different vocabularies, but unify them before merging by assuming that a proposition outside of the scope of a given model evaluates to *maybe* in each state of that model. This interpre-

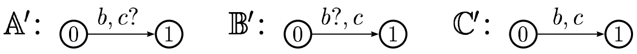

Figure 8.1: Example MTSs for illustrating vocabulary unification.

tation does not preserve the fact that a symbol outside of the vocabulary of a system in our framework cannot cause a change of state in that system, and hence does not ensure that a common observational refinement is constructed. For example, consider models $\mathbb{A}$ and $\mathbb{B}$ in Figure 4.1 with $\alpha\mathbb{A} = \{\mathtt{b}\}$ and $\alpha\mathbb{B} = \{\mathtt{c}\}$, and recall that $\mathbb{C}$, $\mathbb{D}$ and $\mathbb{E}$ are possible merges for these models in our framework. We can use the translation in [18] to obtain the corresponding Kripke Structures for these models and perform vocabulary unification as in [43]. Translating back to MTSs yields models $\mathbb{A}'$ and $\mathbb{B}'$ in Figure 8.1. Clearly, the semantics of these models is not equivalent to models $\mathbb{A}$ and $\mathbb{B}$, for example, because action $\mathtt{c}$ may cause a change of state in model $\mathbb{A}'$. In particular, the merge of these two models is $\mathbb{C}'$, which is not a common refinement of $\mathbb{A}$ and $\mathbb{B}$, since $\mathbb{C}'@\{\mathtt{b}\}$ can internally transit to a state that proscribes $\mathtt{b}$, but $\mathbb{A}$ cannot. Finally, the goal of Nejati and Chechik's work is to provide a framework that supports a negotiation process for inconsistency resolution, intended to help users identify and prioritize disagreements through visualization. In particular, 4-valued Kripke structures are intended for expressing inconsistencies, whereas 3-valued models are not.

MTSs are defined over flat state spaces: Δ^r and Δ^p give a partial description of the behaviours over a *finite* set of states. Huth et al. [28] use the mixed powerdomain of Gunter [20] to generalize MTSs to non-flat state spaces, modelled as domains. Elements of the mixed powerdomain are (roughly) pairs (L, U), where L and U are "special" sets that satisfy a consistency condition, which is a generalization of $L \subseteq U$. The analogy to MTSs is that a state in an MTS should be characterized by the set of properties that are *true* in that state (i.e., L), and the set of properties that are *possible* in that state (i.e., U), and these sets should be consistent in some way (i.e., $L \subseteq U$). Furthermore, there is a natural ordering $\leq$ over the mixed powerdomain that generalizes the notion of refinement of MTSs: $(L, U) \leq (L', U')$ if and only if $L \subseteq L'$ and $U' \subseteq U$. Intuitively, all *true* properties in state (L, U) are *true* in state (L', U') (i.e., $L \subseteq L'$), and all *possible* properties in state (L', U') are *possible* in state (L, U) (i.e., $U' \subseteq U$). In addition, an embedding of standard flat MTSs (and a subset of Mixed Transition Systems [11]) into this domain-theoretic framework is given. Hence, it is conceivable that some results in this thesis can be expressed quite succinctly in terms of the order structure of the mixed powerdomain (e.g., Theorem 3). However, as our results are intended to be explicitly understood by modellers, we have so far refrained from the analogy to mixed powerdomains in order to maintain practicality, and may explore this connection further in future work.

In this thesis, the states that can be merged are those that satisfy consistent sets of temporal properties. However, these assumptions are not universal to all merge frameworks. Finkelstein et al. [15] use first-order logic to express consistency rules, and the models being merged must be consistent prior to merging. Other approaches support merging inconsistent and incomplete views, i.e., enable reasoning in the presence of inconsistencies [14, 45]. Chechik and Easterbrook [14] assume that only states with the same label can be merged, and a similar consistency assumption is made in [54] in the context of UML differencing. Easterbrook and Sabetzadeh [45], on the other hand, give a more general category-theoretic approach based on the observation that it is not always clear how to relate two views. They use graph morphisms to express such relationships, enabling the user to provide this as a third argument to merge. However, their emphasis is on preserving model structure (e.g., states and the accessibility relation between them), rather than on *behavioural* properties.

The operation of merging also arises in several other related areas, including synthesis of State Chart models from scenarios [53, 22], program integration [25], and combining program summaries for software model-checking [2].

Abstraction and Property Preservation. Explicit partiality corresponds naturally to the lack of information at modelling time. Our work has focused on finding a more elaborate model, based on refinement, that preserves the properties of two consistent partial models. The reverse of this process is abstraction, in which a less refined model is constructed. Unlike merge, abstract models are usually hidden from the user for use in automatic procedures, e.g., for efficient model-checking of large or infinite state systems. In addition, the notion of consistency is irrelevant in abstraction, as there is always a model that refines an abstraction, namely the original model itself. However, like merge, soundness of abstractions with respect to property preservation is of fundamental importance in order for abstractions to be of any use when checking properties.

The approach of extending transition systems with a second transition relation describing unknown behaviour was originally proposed by [37], and independently by [11]. Larsen and Thomsen introduced MTSs as a solution to the completeness limitation of LTSs, and proved that Hennessy-Milner logic [23] characterizes strong refinement. Dams' Mixed Transition Systems [11, 10], which are MTSs that do not assume that all required transitions are possible transitions, are used for abstracting Kripke structures. It is shown that 3-valued CTL* properties are preserved by the refinement preorder between these models [11]. Bruns and Godefroid introduced partial Kripke Structures (PKs) [4], which have a single unlabelled transition relation and 3-valued state propositions. They show that 3-valued CTL defined over PKs characterizes their completeness preorder. Huth et al. [28] introduced Kripke MTSs (KMTSs), which are a state-based version of MTSs. KMTSs have two transition relations, as in an MTS, but rather than labelled transitions, each state is labelled with a set of 3-valued propositions. It is shown that 3-valued μ-calculus characterizes refinement defined over KMTSs, which is used as the basis for a 3-valued framework for program analysis.

When a property evaluates to *maybe* in an abstract model, this model must be further refined (where refinement corresponds to splitting abstract states). Shoham and Grumberg [47] show that even standard methods of refining abstract models (e.g. [17]) are not monotonic with respect to property preservation. They define Generalized KMTSs (GKMTSs), an extension of KMTSs with hyper-transitions, as a solution to this problem, and obtain a monotonic abstraction-refinement framework with respect to 3-valued CTL.

Finally, MTSs, KMTSs, and PKs have the same expressive power [18], and in addition, 4-

valued Kripke structures and Mixed Transition Systems share the same expressive power [21].

8.3 Future Work

In the near future, we plan to work on the efficiency of the algorithms presented here, and implement them in the LTSA tool [41]. We also intend to conduct larger case studies that illustrate model elaboration (i.e., when the least common refinement does not exist). Finally, we plan to work on strengthening the determinacy condition to be a necessary condition with respect to uniqueness of merge, improve $+_{ca}$ to build the GCA, and further study algebraic properties.

Our long-term goal is to provide a sound engineering approach to the development of software systems via automated support for constructing partial behavioural models from scenario-based specifications, merging and elaborating these partial behavioural models, as well as enabling users to choose the desired merge from the set of possible minimal common refinements. In particular, we plan to develop a synthesis algorithm for constructing behavioural models from scenarios and integrate this into our approach to merging partial behavioural models.

Bibliography

[1] M. Abadi and L. Lamport. “The Existence of Refinement Mappings”. *Theoretical Computer Science*, 82(2):253–284, 1991.

[2] T. Ball, V. Levin, and F. Xie. “Automatic Creation of Environment Models via Training”. In *Proceedings of 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’04)*, volume 2988 of *LNCS*, pages 93–107. Springer, 2004.

[3] B. Boem and R. Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Person Education, 2004.

[4] G. Bruns and P. Godefroid. “Model Checking Partial State Spaces with 3-Valued Temporal Logics”. In *Proceedings of Proceedings of 11th International Conference on Computer-Aided Verification (CAV’99)*, volume 1633 of *LNCS*, pages 274–287. Springer, 1999.

[5] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. “Multi-Valued Symbolic Model-Checking”. *ACM Transactions on Software Engineering and Methodology*, 12(4):1–38, October 2003.

[6] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. “Progress on the State Explosion Problem in Model Checking”. In R. Wilhelm, editor, *Informatics. 10 Years Back. 10 Years Ahead*, volume 2000 of *LNCS*, pages 176–194. Springer-Verlag, 2001.

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

[8] E.M. Clarke and J. Wing. "Formal Methods: State of the Art and Future Directions". *ACM Computing Surveys*, 28(4):626–643, December 1996.

[9] Rance Cleaveland, Joachim Parrow, and Bernhard 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.

[10] D. Dams, R. Gerth, and O. Grumberg. "Abstract Interpretation of Reactive Systems". *ACM Transactions on Programming Languages and Systems*, 2(19):253–291, 1997.

[11] Dennis Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Eindhoven University of Technology, The Netherlands, July 1996.

[12] R. Diaz-Redondo, J. Pazos-Arias, and A. Fernandez-Vilas. "Reusing Verification Information of Incomplete Specifications". In *Proceedings of the 5th Workshop on Component-Based Software Engineering*, 2002.

[13] A. Dovier, C. Piazza, and A. Policriti. A fast bisimulation algorithm. In *CAV '01: Proceedings of the 13th International Conference on Computer Aided Verification*, pages 79–90, London, UK, 2001. Springer-Verlag.

[14] 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, Toronto, Canada, May 2001. IEEE Computer Society Press.

[15] 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, August 1994.

[16] D. Giannakopoulou and J. Magee. "Fluent Model Checking for Event-Based Systems". In *Proceedings of the 9th joint meeting of the European Software Engineering Con-*

ference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'03), pages 257–266. ACM Press, September 2003.

[17] P. Godefroid, M. Huth, and R. Jagadeesan. "Abstraction-based Model Checking using Modal Transition Systems". In K.G. Larsen and M. Nielsen, editors, *Proceedings of 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *LNCS*, pages 426–440, Aalborg, Denmark, 2001. Springer.

[18] P. Godefroid and R. Jagadeesan. "On the Expressiveness of 3-Valued Models". In *Proceedings of 4th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'03)*, volume 2575 of *LNCS*, pages 206–222. Springer, January 2003.

[19] S. Graf and J. Sifakis. "Readiness Semantics for Regular Processes with Silent Actions". In *14th International Colloquium on Automata, Languages and Programming (ICALP'87)*, pages 115–125. Springer-Verlag, 1987.

[20] C. Gunter. "The Mixed Powerdomain". *Theoretical Computer Science*, 103(2):311–334, 1992.

[21] A. Gurfinkel, O. Wei, and M. Chechik. "Systematic Construction of Abstractions for Model-Checking". In *Proceedings of 7th International Conference on Verification, Model-Checking, and Abstract Interpretation (VMCAI'06)*, volume 3855 of *LNCS*, pages 381–397, Charleston, SC, January 2006. Springer.

[22] D. Harel, H. Kugler, and A. Pnueli. "Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements.". In *Formal Methods in Software and Systems Modeling*, pages 309–324, 2005.

[23] M. Hennessy and R. Milner. "Algebraic Laws for Nondeterminism and Concurrency". *Journal of ACM*, 32(1):137–161, January 1985.

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

[25] S. Horwitz, J. Prins, and T. Reps. “Integrating Noninterfering Versions of Programs.”. *ACM Transactions on Programming Languages and Systems*, 11(3):345–387, 1989.

[26] A. Hunter and B. Nuseibeh. “Managing Inconsistent Specifications: Reasoning, Analysis and Action”. *ACM Transactions on Software Engineering and Methodology*, 7(4):335–367, October 1998.

[27] Altaf Hussain and Michael Huth. “On Model Checking Multiple Hybrid Views”. In *Proceedings of 1st International Symposium on Leveraging Applications of Formal Methods*, pages 235–242, November 2004.

[28] M. Huth, R. Jagadeesan, and D. Schmidt. “A Domain Equation for Refinement of Partial Systems”. Submitted to Mathematical Structures in Computer Science, 2002.

[29] 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.

[30] ITU-T. “ITU-T Recommendation Z.120: Message Sequence Chart (MSC)”. *ITU-T*, 1993.

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

[32] S. C. Kleene. *Introduction to Metamathematics*. New York: Van Nostrand, 1952.

[33] D Kozen. “Results on the Propositional μ-calculus”. *Theoretical Computer Science*, 27:334–354, 1983.

[34] J. Kramer, J. Magee, and M. Sloman. “CONIC: an Integrated Approach to Distributed Computer Control Systems”. *IEE Proceedings*, 130(1):1–10, 1983.

[35] I. Krueger, R. Grosu, P. Scholz, and M. Broy. “From MSCs to Statecharts”. In Franz J. Rammig, editor, *Distributed and Parallel Embedded Systems*. Kluwer Academic Publishers, 1999.

[36] K. Larsen and L. Xinxin. “Equation Solving Using Modal Transition Systems”. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science (LICS’90)*, pages 108–117. IEEE Computer Society Press, 1990.

[37] 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.

[38] Kim G. Larsen, Bernhard Steffen, and Carsten Weise. “A Constraint Oriented Proof Methodology based on Modal Transition Systems”. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS’95)*, LNCS, pages 13–28. Springer, May 1995.

[39] E. Letier, J. Kramer, J. Magee, and S. Uchitel. “Fluent Temporal Logic for Discrete-time Event-Based Models”. In *Proceedings of the 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE’05)*, pages 70–79. ACM Press, 2005.

[40] X. Liu. *“Specification and Decomposition in Concurrency”*. PhD thesis, Department of Computer Science, Aalborg University, 1992.

[41] J. Magee and J. Kramer. *“Concurrency - State Models and Java Programs”*. John Wiley, 1999.

[42] R. Milner. *Communication and Concurrency*. Prentice-Hall, New York, 1989.

[43] 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.

[44] Robert Paige and Robert Tarjan. “Three Partition Refinement Algorithms”. *SIAM Journal of Computing*, 16(6):973–989, December 1987.

[45] 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.

[46] ICSE Workshop on Scenarios and State Machines: Model, Algorithms and Tools (SCESM), 2002-2004.

[47] S. Shoham and O. Grumberg. “Monotonic Abstraction-Refinement for CTL”. In *Proceedings of 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’04)*, volume 2988 of *LNCS*, pages 546–560. Springer-Verlag, April 2004.

[48] Colin 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.

[49] 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.

[50] S. Uchitel, J. Kramer, and J. Magee. “Behaviour Model Elaboration using Partial Labelled Transition 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 19–27, 2003.

[51] S. Uchitel, J. Kramer, and J. Magee. “Synthesis of Behavioural Models from Scenarios”. *IEEE Transactions on Software Engineering (TSE’03)*, 29(2):99–115, 2003.

[52] S. Uchitel, J. Kramer, and J. Magee. “Incremental Elaboration of Scenario-Based Specifications and Behaviour Models using Implied Scenarios”. *ACM Transactions on Software Engineering and Methodology (TOSEM’04)*, 13(1):37–85, 2004.

[53] J. Whittle and J. Schumann. “Generating Statechart Designs from Scenarios”. In *Proceedings of 22nd International Conference on Software Engineering (ICSE’00)*, pages 314–323. ACM Press, May 2000.

[54] Z. Xing and E. Stroulia. “UMLDiff: An Algorithm for Object-Oriented Design Differencing”. In *Proceedings of 20th IEEE International Conference on Automated Software Engineering (ASE’05)*, pages 54–65. IEEE Computer Society, 2005.

Appendix A

Additional Case Study Code

Here we provide additional FSP code for the components in the mine pump case study described in Chapter 7.

A.1 FSP Code for Generating Models

The following FSP code is used to generate the MTSs and explicit state FSP code.

```
/* The Mine Pump Case Study */

const False = 0
const True  = 1
range Bool = False..True

WaterSensor = LOW,
LOW = (medWater -> MIDDLE),
MIDDLE = (lowWater
-> LOW | highWater -> HIGH),
HIGH = (medWater -> MIDDLE).

MethaneSensor = NOTPRESENT,
NOTPRESENT = (methAppears -> PRESENT),
PRESENT = (methLeaves -> NOTPRESENT).

Pump = OFF,
OFF = (switchOn -> ON),
ON = (switchOff -> OFF).

/**********************************************
* Inconsistent Policies that enforce immediacy
**********************************************/

/* without danger light */
OffPolicy = PUMP[True][False][False],
PUMP[lowWater:Bool][methane:Bool][pumpOn:Bool] = (
     // maybes: leave on policy open
    when (!pumpOn) switchOn -> PUMP[lowWater][methane][True] |
    medWater -> PUMP[False][methane][pumpOn] |
    when (pumpOn) lowWater -> switchOff -> PUMP[True][methane][False] |
    when (!pumpOn) lowWater -> PUMP[True][methane][pumpOn] |
    when (pumpOn) methAppears -> switchOff -> PUMP[lowWater][True][False] |
    when (!pumpOn) methAppears -> PUMP[lowWater][True][pumpOn] |
    methLeaves -> PUMP[lowWater][False][pumpOn]
).

/* with danger light */
OffPolicyDL =
PUMP[True][False][False][False],
PUMP[lowWater:Bool][methane:Bool][pumpOn:Bool][dangerLight:Bool] =
(
    lowWater -> PUMP[True][methane][pumpOn][dangerLight] |
    {medWater,highWater} -> PUMP[False][methane][pumpOn][dangerLight] |
    methAppears -> PUMP[lowWater][True][pumpOn][dangerLight] |
    methLeaves -> PUMP[lowWater][False][pumpOn][dangerLight] |
    // off policy: switch off at lowWater
    when (pumpOn) lowWater -> switchOff -> PUMP[lowWater][methane][False][dangerLight] |
    // maybes: leave on policy open
    when (!pumpOn) switchOn -> PUMP[lowWater][methane][True][dangerLight] |
    when (!dangerLight && pumpOn) methAppears -> dangerLightOn
    -> switchOff -> PUMP[lowWater][True][False][True] |
    when (!dangerLight && !pumpOn) methAppears -> dangerLightOn
    -> PUMP[lowWater][True][False][True] |
    when (dangerLight) methLeaves -> dangerLightOff -> PUMP[lowWater][False][pumpOn][False]
).

OnPolicy = PUMP[False][False][False],
PUMP[highWater:Bool][methane:Bool][pumpOn:Bool] = (
    medWater -> PUMP[False][methane][pumpOn] |
    // maybes: might turn it on at medWater (we do not enforce immediacy here)
    when (!pumpOn && !methane && !highWater) switchOn -> PUMP[highWater][methane][True] |
    // on policy: enforces immediacy
    when (!pumpOn && !methane) highWater -> switchOn -> PUMP[True][methane][True] |
    when (pumpOn || methane) highWater -> PUMP[True][methane][pumpOn] |
    // maybes: leave off policy open
    when (pumpOn) switchOff -> PUMP[highWater][methane][False] |
    methAppears -> PUMP[highWater][True][pumpOn] |
    when (highWater && !pumpOn) methLeaves -> switchOn -> PUMP[highWater][False][True] |
    when (!highWater || pumpOn) methLeaves -> PUMP[highWater][False][pumpOn]
).

||MinePump1 = (OnPolicy || WaterLevelSensor || MethaneSensor || Pump).
||MinePump2 = (OffPolicyDL || WaterLevelSensor || MethaneSensor || Pump). ).


/*****************************************************************
* Consistent timed policies that do not enforce strict immediacy
*****************************************************************/

/* without danger light */
OffPolicyNoIm = PUMP[True][False][False],
PUMP[lowWater:Bool][methane:Bool][pumpOn:Bool] = (
    lowWater -> PUMP[True][methane][pumpOn] |
    {medWater,highWater} -> PUMP[False][methane][pumpOn] |
    methAppears -> PUMP[lowWater][True][pumpOn] |
    methLeaves -> PUMP[lowWater][False][pumpOn] |
    // off policy: switch off at lowWater
    when (pumpOn && lowWater) switchOff -> PUMP[lowWater][methane][False] |
    // off policy: switch off if methAppears
    when (pumpOn && methane) switchOff -> PUMP[lowWater][methane][False] |
    // maybes: leave on policy open
    when (!pumpOn) switchOn -> PUMP[lowWater][methane][True] |
    // maybe ticks
    when (!(methane || lowWater) || !pumpOn) tick -> PUMP[lowWater][methane][pumpOn]
).
```

```
/* with danger light */
OffPolicyNoImDL = PUMP[True][False][False][False],
PUMP[lowWater:Bool][methane:Bool][pumpOn:Bool][dangerLight:Bool] = (
    lowWater -> PUMP[True][methane][pumpOn][dangerLight] |
    {medWater,highWater} -> PUMP[False][methane][pumpOn][dangerLight] |
    methAppears -> PUMP[lowWater][True][pumpOn][dangerLight] |
    methLeaves -> PUMP[lowWater][False][pumpOn][dangerLight] |
    // off policy: switch off at lowWater
    when (pumpOn && lowWater) switchOff -> PUMP[lowWater][methane][False][dangerLight] |
    // off policy: switch off if methAppears
    when (pumpOn && methane) switchOff -> PUMP[lowWater][methane][False][dangerLight] |
    // maybes: leave on policy open
    when (!pumpOn) switchOn -> PUMP[lowWater][methane][True][dangerLight] |
    when (!dangerLight && methane) dangerLightOn -> PUMP[lowWater][True][pumpOn][True] |
    when (dangerLight && !methane) dangerLightOff -> PUMP[lowWater][False][pumpOn][False] |
    // maybe ticks
    when (!(methane || lowWater) || !pumpOn) tick
    -> PUMP[lowWater][methane][pumpOn][dangerLight]
).

OnPolicyNoIm = PUMP[False][False][False],
PUMP[highWater:Bool][methane:Bool][pumpOn:Bool] = (
    {lowWater,medWater} -> PUMP[False][methane][pumpOn] |
    highWater -> PUMP[True][methane][pumpOn] |
    methAppears -> PUMP[highWater][True][pumpOn] |
    methLeaves -> PUMP[highWater][False][pumpOn] |
    // maybes: might turn it on at low or medium water
    when (!pumpOn && !methane && !highWater) switchOn -> PUMP[highWater][methane][True] |
    // on policy: turn on when highWater, pump off, and no methane
    when (!pumpOn && !methane && highWater) switchOn -> PUMP[highWater][methane][True] |
    // maybes: leave off policy open
    when (pumpOn) switchOff -> PUMP[highWater][methane][False] |
     // maybe ticks
    when (!(highWater && !methane) || pumpOn) tick -> PUMP[highWater][methane][pumpOn]
).

||MinePumpNoIm1 = (OnPolicyNoIm || WaterLevelSensor || MethaneSensor || Pump).
||MinePumpNoIm2 = (OffPolicyNoImDL || WaterLevelSensor || MethaneSensor || Pump).
```

A.2 FSP Code for Generated Models

Here we provide explicit state FSP code for the components omitted from Chapter 7, generated by the code found in Section A.1 above. Because FSP cannot explicitly handle maybe transitions, they must be added in manually.

```
MinePump1 = Q0,
Q0  = (methAppears -> Q1
      |switchOn? -> Q11
      |medWater -> Q12),
Q1  = (methLeaves -> Q0
      |medWater -> Q2),
Q2  = (lowWater -> Q1
      |highWater -> Q3
      |methLeaves -> Q12),
Q3  = (medWater -> Q2
      |methLeaves -> Q4),
Q4  = (switchOn -> Q5),
Q5  = (switchOff? -> Q6
      |methAppears -> Q7
      |medWater -> Q10),
Q6  = (methAppears -> Q3
      |medWater -> Q12),
Q7  = (switchOff? -> Q3
      |methLeaves -> Q5
      |medWater -> Q8),
Q8  = (switchOff? -> Q2
      |highWater -> Q7
      |lowWater -> Q9
      |methLeaves -> Q10),
Q9  = (switchOff? -> Q1
      |medWater -> Q8
      |methLeaves -> Q11),
Q10 = (highWater -> Q5
      |methAppears -> Q8
      |lowWater -> Q11
      |switchOff? -> Q12),
Q11 = (switchOff? -> Q0
      |methAppears -> Q9
      |medWater -> Q10),
Q12 = (lowWater -> Q0
      |methAppears -> Q2
      |highWater -> Q4
      |switchOn? -> Q10).
```

```
MinePump2 = Q0
Q0  = (switchOn? -> Q1                   Q11 = (methLeaves -> Q0
      |methAppears -> Q11                     |switchOn? -> Q8
      |methAppears -> Q12                     |medWater -> Q13),
      |medWater -> Q20),                Q12 = (dangerLightOn -> Q4),
Q1  = (methAppears -> Q2                Q13 = (switchOn? -> Q9
      |methAppears -> Q8                      |{highWater, lowWater} -> Q11
      |medWater -> Q16),                      |methLeaves -> Q20),
Q2  = (dangerLightOn -> Q3),            Q14 = ({highWater, lowWater} -> Q4
Q3  = (switchOff -> Q4),                      |methLeaves -> Q7
Q4  = (methLeaves -> Q5                       |methLeaves -> Q15),
      |methLeaves -> Q6                 Q15 = (dangerLightOff -> Q20),
      |medWater -> Q14),                Q16 = ({highWater, lowWater} -> Q1
Q5  = (dangerLightOff -> Q0),                 |methAppears -> Q9
Q6  = (methAppears -> Q4                      |methAppears -> Q17
      |medWater -> Q7),                       |lowWater -> Q19),
Q7  = ({highWater, lowWater} -> Q6      Q17 = (dangerLightOn -> Q18),
      |methAppears -> Q14),             Q18 = (switchOff -> Q14),
Q8  = (methLeaves -> Q1                 Q19 = (switchOff -> Q0),
      |medWater -> Q9),                 Q20 = ({highWater, lowWater} -> Q0
Q9  = ({highWater, lowWater} -> Q8            |methAppears -> Q13
      |lowWater -> Q10                        |switchOn? -> Q16
      |methLeaves -> Q16),                    |methAppears -> Q21),
Q10 = (switchOff -> Q11),               Q21 = (dangerLightOn -> Q14).
```

```
MinePumpNoIm1 = Q0,
Q0  = (tick? -> Q0          Q4  = (tick? -> Q4                 |tick? -> Q8),
      |switchOn? -> Q1            |highWater -> Q5       Q9  = (lowWater -> Q1
      |methAppears -> Q10         |lowWater -> Q10             |methAppears -> Q3
      |medWater -> Q11),          |methLeaves -> Q11),         |highWater -> Q7
Q1  = (switchOff? -> Q0     Q5  = (medWater -> Q4              |tick? -> Q9
      |tick? -> Q1                |tick? -> Q5                 |switchOff? -> Q11),
      |methAppears -> Q2          |methLeaves -> Q6),    Q10 = (methLeaves -> Q0
      |medWater -> Q9),     Q6  = (methAppears -> Q5           |medWater -> Q4
Q2  = (methLeaves -> Q1           |switchOn -> Q7              |tick? -> Q10),
      |tick? -> Q2                |medWater -> Q11),     Q11 = (lowWater -> Q0
      |medWater -> Q3       Q7  = (switchOff? -> Q6            |methAppears -> Q4
      |switchOff? -> Q10),        |tick? -> Q7                 |highWater -> Q6
Q3  = (lowWater -> Q2             |methAppears -> Q8           |switchOn? -> Q9
      |tick? -> Q3                |medWater -> Q9),            |tick? -> Q11).
      |switchOff? -> Q4     Q8  = (medWater -> Q3
      |highWater -> Q8            |switchOff? -> Q5
      |methLeaves -> Q9),         |methLeaves -> Q7
```

```
MinePumpNoIm2 = Q0,
Q0  = (tick? -> Q0               Q8  = (medWater -> Q7               |switchOn? -> Q8
      |switchOn? -> Q1                 |tick? -> Q8                  |methAppears -> Q12
      |methAppears -> Q22              |dangerLightOff -> Q9         |dangerLightOff -> Q13
      |medWater -> Q23),               |methAppears -> Q15),         |tick? -> Q16),
Q1  = (switchOff -> Q0           Q9  = (tick? -> Q9            Q17 = (lowWater -> Q4
      |methAppears -> Q2               |methAppears -> Q10           |methLeaves -> Q6
      |medWater -> Q21),               |medWater -> Q21),            |highWater -> Q12
Q2  = (methLeaves -> Q1          Q10 = (methLeaves -> Q9             |tick? -> Q17
      |dangerLightOn -> Q3             |switchOff -> Q11             |switchOn? -> Q19),
      |medWater -> Q20                 |dangerLightOn -> Q15   Q18 = (dangerLightOff -> Q1
      |switchOff -> Q22),              |medWater -> Q20),            |methAppears -> Q3
Q3  = (switchOff -> Q4           Q11 = (switchOn? -> Q10             |switchOff -> Q5
      |methLeaves -> Q18               |tick? -> Q11                 |medWater -> Q7),
      |medWater -> Q19),               |dangerLightOn -> Q12   Q19 = (lowWater -> Q3
Q4  = (switchOn? -> Q3                 |methLeaves -> Q13            |methLeaves -> Q7
      |tick? -> Q4                     |medWater -> Q14),            |highWater -> Q15
      |methLeaves -> Q5          Q12 = (tick? -> Q12                 |switchOff -> Q17),
      |medWater -> Q17),               |switchOn? -> Q15       Q20 = (lowWater -> Q2
Q5  = (dangerLightOff -> Q0            |methLeaves -> Q16            |highWater -> Q10
      |methAppears -> Q4               |medWater -> Q17),            |switchOff -> Q14
      |tick? -> Q5               Q13 = (switchOn? -> Q9              |dangerLightOn -> Q19
      |medWater -> Q6                  |methAppears -> Q11          |methLeaves -> Q21),
      |switchOn? -> Q18),
Q6  = (lowWater -> Q5
      |tick? -> Q6
      |switchOn? -> Q7
      |highWater -> Q16
      |methAppears -> Q17
      |dangerLightOff -> Q23),
Q7  = (tick? -> Q7
      |highWater -> Q8
      |lowWater -> Q18
      |methAppears -> Q19
      |dangerLightOff -> Q21),
      |tick? -> Q13
      |medWater -> Q23),
Q14 = (highWater -> Q11
      |tick? -> Q14
      |dangerLightOn -> Q17
      |switchOn? -> Q20
      |lowWater -> Q22
      |methLeaves -> Q23),
Q15 = (methLeaves -> Q8
      |switchOff -> Q12
      |medWater -> Q19),
Q16 = (medWater -> Q6
Q21 = (lowWater -> Q1
      |highWater -> Q9
      |methAppears -> Q20
      |tick? -> Q21),
Q22 = (methLeaves -> Q0
      |switchOn? -> Q2
      |dangerLightOn -> Q4
      |medWater -> Q14
      |tick? -> Q22),
Q23 = (lowWater -> Q0
      |highWater -> Q13
      |methAppears -> Q14
      |switchOn? -> Q21
      |tick? -> Q23).
```