Let's Agree to Disagree

Shiva Nejati Marsha Chechik
Department of Computer Science
University of Toronto
40 St. George Street
Toronto, Ontario, Canada M5S 2E4
Email: {shiva,chechik}@cs.toronto.edu

ABSTRACT

Almost every kind of software development periodically needs to merge models. Perhaps they come from different stakeholders during the requirements analysis phase, or perhaps they are modifications of the same model done independently by several groups of people. Sometimes these models are consistent and can be merged. Sometimes they are not, and negotiation between the stakeholders is needed in order to resolve inconsistencies. While various methods support merging, we need formal approaches that help stakeholders negotiate.

In this paper, we present a formal framework for merging and conflict resolution. It facilitates automatic merging of consistent models, enables users to visualize and explore potential disagreements and identify their priorities, and suggests ways to resolve the priority items. We describe our implementation of the framework and illustrate it on several examples.

Categories and Subject Descriptors: D.2.1 [Software Engineering]: Requirements/Specifications; D.2.4 Software/Program Verification.

Keywords: Model Merging, Refinement, Model Checking, Inconsistency Detection, Negotiation, 3-Valued Logic.

1. INTRODUCTION

Almost every kind of software development periodically needs to merge models. For example, during requirements analysis, different stakeholders with different viewpoints [35] describe different, yet overlapping aspects [7] of the same systems. 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 [39]. Standard approaches to synthesis produce a separate behavioural model for each client instance (e.g., [40, 30]). It is reasonable to integrate all models of all client instances into a single model for the client component type because all clients should share the same characteristics.

The problem gets even more pressing when we are dealing with distributed software development [10], when teams in different locations independently modify a common model, and then attempt to put their modifications together.

In this paper, we concentrate on merging behavioural models of software. There are several ways to express such models; these are typically divided into declarative specifications, such as Alloy [27], and operational specifications, expressed in some form of state-machines. State-machines are widely used in requirements modeling [20, 21, 22] either directly, or via translation from higher-level modeling languages.

In the context of model elaboration, composition of two (partial) descriptions of the *same* component to obtain a more elaborate version of the original partial description has been called *merge* [39]. Effective merging supports collaboration and cooperation in the process of specifying software and helps manage the complexities of this process. Unfortunately, merging can combine models only if there are no disagreements between the stakeholders. Otherwise, this composition requires *negotiation*. In order to support state-machine-based development, we need to be able to merge different versions of state-machines as well as support possible conflict resolution.

Merging and negotiation go hand in hand, and we believe that this process should be supported by a formal framework. Such a framework should merge models, if they are consistent, and otherwise support negotiation by helping users discover their disagreements, allow them to trace through their decisions and understand proposals with the goal of resolving conflicts. This paper presents such a framework in the context of state-machine models. We assume that each entity in our models is specified at the same level of abstraction and is named consistently in each model, i.e., we assume *vocabulary consistency*.

Formal support for model-merging has been addressed by several researchers. For example, merging is just a conjunction of the corresponding theories in declarative specifications [27]. Uchitel and Chechik [39] define merging for consistent partial labelled transition systems, and Huth and Pradhan [26] merge partial view-based specifications where a dominance ordering is used to eliminate the potential inconsistencies. Different aspects of negotiation have been addressed in software engineering literature. For example, [2] describes negotiation over non-functional and application-independent goals, such as the trade-off between assurance and performance or cost/schedule. Damian et. al. [10] consider social and political aspects of negotiation, and [9, 12] take a dialectic reasoning approach to negotiation. Several researchers [13, 23] proposed ways to do formal reasoning with inconsistency; however, we are not aware of formal support for *negotiation* over inconsistent behavioural models.

Given two models, our framework automatically determines whe-

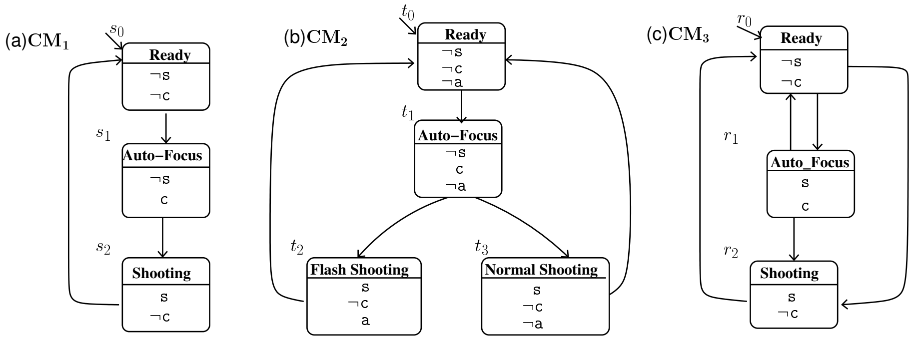

Figure 1: Models of the photo-taking feature of a camera: (a) $\mathbf{CM_1}$; (b) $\mathbf{CM_2}$; (c) $\mathbf{CM_3}$.

ther the models can be merged, and if so, computes the merge. Otherwise, it supports the negotiation process, helping users identify their disagreements and prioritize them. Further, it provides automated support for computing proposals for models that bring users closer to resolving their conflicts, allowing users to do "what if" exploration and choose the most suitable alternatives. We also keep a history of decisions that have been made, allowing users to study the results, and, if necessary, undo the decisions. Our methodology also guarantees that the negotiation process will eventually terminate, while inflicting only the minimal changes onto the original models.

In this paper, we describe the framework, analyze its complexity, and illustrate that it scales to non-trivial models. Like the work in [39, 26], we use additional logic values to capture model incompleteness. Multi-valued logic has also been recently used for *reasoning* [23, 13, 4, 15] about inconsistent systems, with the goal of determining which inconsistencies can be tolerated. In our approach, we never merge inconsistent systems, and use the exploration phase to determine those inconsistencies that need to be resolved, and those that can be tolerated.

The rest of this paper is organized as follows. In Section 2, we introduce a running example. In Section 3, we identify a class of consistent models and show how to merge them. Section 4 presents the main results of this paper: techniques to cope with inconsistency through exploration and selection of suitable resolutions. We discuss the implementation of our framework and an additional case study in Section 5, and look at alternatives for improving precision of our analysis in Section 6. Section 7 compares our approach to related work, whereas Section 8 summarizes the paper and outlines venues for future work. Appendix A provides proofs of the theorems that appear in the paper.

2. EXAMPLE

We illustrate our framework on several specification models of the photo-taking feature of a camera[1]. To take a photo, a user needs to press the *shutter* button half-way. When *focus* is achieved, the shutter button can be pressed completely to take the picture. Under low-light conditions, the built-in *flash* should fire automatically.

Three different specification models of a camera, $\mathbf{CM_1}$, $\mathbf{CM_2}$, and $\mathbf{CM_3}$ are shown in Figure 1. The goal of $\mathbf{CM_1}$ is to specify the focusing feature and the behaviour of the camera's shutter. In the first state of this model, the shutter is closed and the focus is not yet achieved; in the second, the focus is achieved; and in the third, the shutter becomes open so that the photo can be taken. Model $\mathbf{CM_2}$ (see Figure 1(b)) additionally describes the built-in flash. It is disabled in the first and second states; in the third, the camera opens its shutter and, depending on the light intensity, the flash is either fired or remains disabled.

Like $\mathbf{CM_1}$, model $\mathbf{CM_3}$, shown in Figure 1(c), only considers focusing and the camera's shutter. However, unlike $\mathbf{CM_1}$, $\mathbf{CM_3}$ assumes that there is a transition from state **Ready** to state **Shooting**, i.e., the camera can take a photo even without achieving focus. Further, when focus is achieved, $\mathbf{CM_3}$ allows a user to avoid taking a photo. This is indicated by a transition from **Auto-Focus** to **Ready**. Finally, $\mathbf{CM_3}$ (mistakenly) allows the shutter to be open during focusing, i.e., in state **Auto_Focus**.

In the camera example, we assume vocabulary consistency: all stakeholders use a, c, and s to represent whether the flash is enabled, whether the focus is achieved, and whether the camera's shutter is open, respectively. We refer to the set of variables used by a model as its *context* (e.g., $\{\mathtt{c}, \mathtt{s}\}$ for $\mathbf{CM_1}$ and $\mathbf{CM_3}$), and the union of all contexts as the *unified set of variables*. In general, achieving and maintaining vocabulary consistency is a difficult problem, studied, e.g., by [17]. We consider this issue to be orthogonal to the techniques presented in this paper.

State-machine models are typically constructed to ensure that the resulting design satisfies (or violates) certain properties. For example, some properties of the camera example are shown in Table 1. These properties are either representations of individual executions of the system, such as *use cases* or *scenarios* (e.g., $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_4}$), or statements about *all* system executions, such as invariants (e.g., $\mathbf{P_3}$ and $\mathbf{P_5}$). $\mathbf{P_1}$ and $\mathbf{P_2}$ are positive scenarios whereas $\mathbf{P_4}$ is a negative scenario: it prohibits behaviours where **Shooting** is immediately followed by **Ready**. We can easily show that $\mathbf{CM_1}$ and $\mathbf{CM_2}$ satisfy $\mathbf{P_1}$, $\mathbf{P_3}$ and $\mathbf{P_4}$, and violate $\mathbf{P_2}$, whereas $\mathbf{CM_3}$ satisfies $\mathbf{P_1}$ and $\mathbf{P_2}$, and violates $\mathbf{P_3}$ and $\mathbf{P_4}$. If we think of $\mathbf{P_1}$–$\mathbf{P_4}$ as the desirable properties to be achieved by the combined camera model, $\mathbf{CM_1}$ and $\mathbf{CM_2}$ disagree with $\mathbf{CM_3}$ on these. Thus, we expect that $\mathbf{CM_3}$ cannot be merged with $\mathbf{CM_1}$ and $\mathbf{CM_2}$ without some negotiation about $\mathbf{P_2}$, $\mathbf{P_3}$ and $\mathbf{P_4}$.

When present, global properties may give analysts an early warning about whether some negotiation would be required. Of course, it is possible that all models agree on the specified global properties and still cannot be merged without negotiation (i.e., due to requirements that have not been stated explicitly), or that global properties are not present at all. Whatever is the case, our framework attempts to merge models, and assists users in resolving possible inconsistencies.

We denote the set $\{\mathtt{c}, \mathtt{a}, \mathtt{s}\}$, the unified set of variables for the camera model, by AP_u. We also omit state names since our modelling formalism completely characterizes each state by values of its variables.

3. MERGING CONSISTENT MODELS

In this section, we look at the problem of merging consistent

[1]The example is adapted from [37].

Property	Description	CTL formulation
$\mathbf{P_1}$	whenever focus is achieved, we can take a picture	$AG(\mathsf{c} \Rightarrow EX\mathsf{s})$
$\mathbf{P_2}$	we can achieve focus and yet not take a picture	$EF(\mathsf{c} \Rightarrow EX\neg\mathsf{s})$
$\mathbf{P_3}$	when focus is being achieved, a picture cannot be taken	$AG(\mathsf{c} \Rightarrow \neg\mathsf{s})$
$\mathbf{P_4}$	we cannot take a picture without achieving focus	$\neg E[\neg\mathsf{c}\ U\ \mathsf{s}]$
$\mathbf{P_5}$	whenever flash is enabled, shutter is open	$AG(\mathsf{a} \Rightarrow \mathsf{s})$

Table 1: Properties of the camera models.

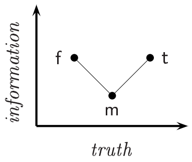

Figure 2: 3-valued logic.

models. Section 3.1 gives some background information, and Section 3.2 describes which models can be merged and how to do it. Like Uchitel and Chechik [39], we define merge as refinement of two models; however, they merge labelled transition systems, whereas we consider state-based models.

3.1 Basic Notions

Requirements models are inherently incomplete. Each model can only focus on a few features of a system and thus uses just a fraction of the unified set of variables. For example, $\mathbf{CM_1}$ (see Figure 1(a)) does not address the built-in flash feature and thus does not use the variable a. We can also say that in any state of $\mathbf{CM_1}$, the value of a is unspecified.

To be able to merge models, we need to unify contexts of these models and address the resulting incompleteness. We do so using 3-valued logic. *3-valued logic* [28], shown in Figure 2, has been recently used by several researchers to model and reason with incompleteness and uncertainty (e.g., [24, 26, 39, 3, 4]). It extends classical logic with an additional truth value, denoted by maybe (m). For example, when the context of $\mathbf{CM_1}$ is lifted to AP_u, we simply set the missing variable a to m in all of the states of $\mathbf{CM_1}$. The result is shown in Figure 3(a).

3-valued logic provides a suitable means to express the degree of information via *information ordering* (denoted by an operator $\preceq$) over its truth values. It is shown in Figure 2: true (t) and false (f) are more defined than m and incomparable with each other. For example, variable a in state s_1 of model $\mathbf{A}$ (see Figure 3(a)) is less defined than a in state s_1 of model $\mathbf{B}$ (see Figure 3(b)). We denote the set $\{\mathsf{t}, \mathsf{m}, \mathsf{f}\}$ by $\mathbf{3}$. We also define the meet and the join operators with respect to $\preceq$, denoting them by $\sqcap$ and $\sqcup$, respectively. For example, $\mathsf{m} \sqcap \mathsf{t} = \mathsf{m}$. Note that $\mathsf{t} \sqcup \mathsf{f}$ is not defined.

We define a *truth ordering* over 3-valued truth values, denoted by $\leq$ and shown in Figure 2, where $\mathsf{f} \leq \mathsf{m} \leq \mathsf{t}$. Like many other researchers, e.g., [4, 25], we use the truth ordering to evaluate properties over models. For example, $\mathbf{P_3}$ (see Table 1) indicates that c and s cannot be true at the same time. In state r_1 of model $\mathbf{F}$ in Figure 3(f), $\mathsf{c} = \mathsf{t}$ and $\mathsf{s} = \mathsf{m}$; thus, $\mathbf{P_3}$ evaluates to m in r_1 and thus in $\mathbf{F}$.

We may allow our models to include 3-valued transitions as well. For example, Figure 3(b) shows a fragment of a camera model where transitions from s_0 to s_1 and s_1' are m, which represents the fact that this model is not sure about the status of the camera's flash (a) after the initial state s_0. From the theoretical point of view, adding m transitions to models with m variables does not add any expressive power [16], but it is often convenient. In our examples, we use the following convention for transitions: unlabelled transitions have value t, transitions labelled with m have value m, and f transitions are not shown.

We define our models to be tuples (Σ, s_0, R, I, AP), where Σ is a set of states, $s_0 \in \Sigma$ is an initial state, $R : (\Sigma \times \Sigma) \to \mathbf{3}$ is a transition function, $AP \subseteq AP_u$ is a set of atomic propositions (variables), and $I : (\Sigma \times AP) \to \mathbf{3}$ is an interpretation function that determines the value of each variable at every state. Without loss of generality, we assume that state machines have only one initial state.

To compare models with different variable sets, we assume that the labelling function for every state machine is defined for every variable in AP_u, and further, for every $p \in AP_u \setminus AP$ and every $s \in \Sigma$, $I(s, p) \triangleq \mathsf{m}$.

We now extend the information ordering $\preceq$ to 3-valued state machines, resulting in *refinement*.

DEFINITION 1. (refinement) *[25, 31] Let 3-valued state machines $M_1 = (\Sigma_1, s_0, R_1, I_1, AP_1)$ and $M_2 = (\Sigma_2, t_0, R_2, I_2, AP_2)$ be given. A relation $\preceq\ \subseteq \Sigma_1 \times \Sigma_2$ is a **refinement** where $s \preceq t$ iff*
1. $\forall p \in AP_u \cdot I_1(s,p) \preceq I_2(t,p)$
2. $\forall s' \in \Sigma_1 \cdot R_1(s,s') \succeq \mathsf{t} \;\Rightarrow\; \exists t' \in \Sigma_2 \cdot R_2(t,t') \succeq \mathsf{t} \;\wedge\; s' \preceq t'$
3. $\forall t' \in \Sigma_2 \cdot R_2(t,t') \preceq \mathsf{t} \;\Rightarrow\; \exists s' \in \Sigma_1 \cdot R_1(s,s') \preceq \mathsf{t} \;\wedge\; s' \preceq t'$

We say M_2 refines M_1, written as $M_1 \preceq M_2$, iff $s_0 \preceq t_0$.

Intuitively, t refines s if the variables in s are less defined than those in t (condition 1), every definite, i.e., t, transition from s is matched by some definite, i.e., t, transition from t (condition 2), and every possible, i.e., t or m, transition from t is matched by a possible, i.e., t or m, transition from s (condition 3). M_2 refines M_1 if the guaranteed behaviors of M_1 are a subset of the guaranteed behaviors of M_2; and the possible behaviors of M_2 are a subset of the possible behaviors of M_1. For example, model $\mathbf{C} = (\Sigma_\mathbf{c}, ...)$, shown in Figure 3(c), is a refinement of model $\mathbf{A}$ (see Figure 3(a)), where the refinement relation is $\{(s, (s, x)) \mid (s, x) \in \Sigma_\mathbf{c}\}$.

Note that our treatment of the labelling function allows us to compare models with different variable sets.

Refinement preserves all definite behaviours of the original model. Furthermore, it preserves truth and falsity of properties expressed in the temporal logic L_μ (μ-calculus) [25]. L_μ[29] is a very powerful logic, capable of capturing a wide variety of global properties: traces, scenarios, invariants (e.g., properties $\mathbf{P_1}$–$\mathbf{P_5}$ introduced in Section 2) and many more. The 3-valued semantics of L_μ is given in [18]. In this paper, we use *computational tree logic* (CTL) [8], a subset of μ-calculus, to formalize our properties. CTL is a branching-time temporal logic defined by the following grammar:

$$\begin{array}{rcl}\varphi & = & \ell \mid p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid EX\varphi \mid AX\varphi \mid EF\varphi \\ & & \mid AF\varphi \mid EG\varphi \mid AG\varphi \mid E[\varphi\ U\ \varphi] \mid A[\varphi\ U\ \varphi]\end{array}$$

where $p \in AP$ is an atomic proposition and $\ell \in \mathbf{3}$. The meaning of the temporal operators is: given a state and paths emanating from it, φ holds in one (EX) or all (AX) next states; φ holds in some future state along one (EF) or all (AF) paths; φ holds globally along one (EG) or all (AG) paths, and φ holds until a point where ψ holds along one (EU) or all (AU) paths.

We write $||\varphi||^M(s)$ to indicate the value of φ in the state s of M, and $||\varphi||(s)$ when M is clear from the context. The value of

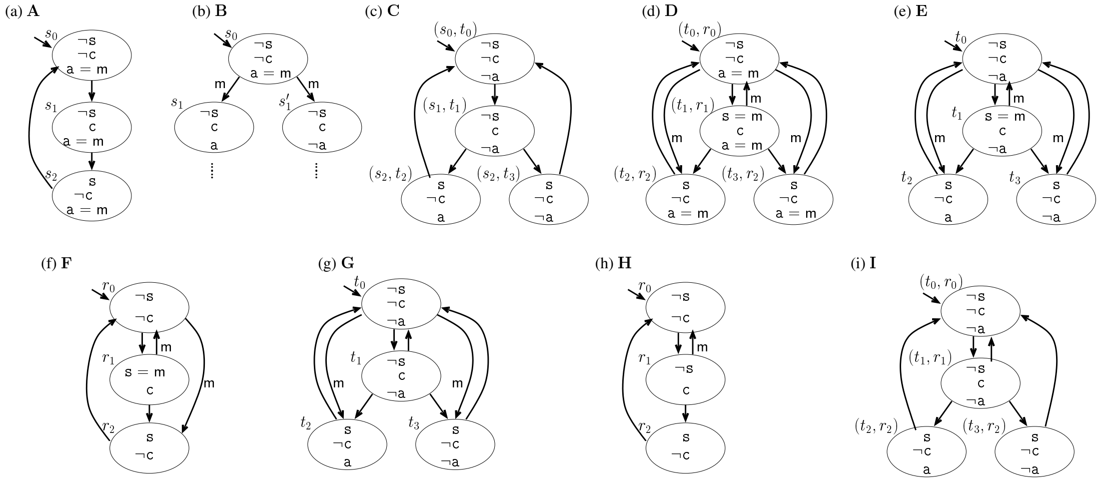

Figure 3: Example models: (a) $\mathbf{CM_1}$ with the context $\{\mathsf{s}, \mathsf{c}, \mathsf{a}\}$; (b) a model with 3-valued transitions; (c) $\mathbf{CM_1} + \mathbf{CM_2}$; (d) $\mathbf{CM_2} \oplus \mathbf{CM_3}$; (e) a projection of (d) onto $\mathbf{CM_2}$; (f) a projection of (d) onto $\mathbf{CM_3}$; (g) a proposal for modifying $\mathbf{CM_2}$; (h) a proposal for $\mathbf{CM_3}$; and (i) a merge of (g) and (h).

the formula φ in a 3-valued state machine M is its value in the initial state, i.e., $||\varphi||^M \triangleq ||\varphi||^M(s_0)$. Temporal operators EX, EG, and EU together with the propositional connectives form an adequate set (i.e., all other operators can be defined from them). For example, $EF\varphi \triangleq E[\mathsf{t}\ U\ \varphi]$ and $AG\varphi \triangleq \neg EF \neg\varphi$.

A path emanating from a state s of a 3-valued state machine M is a sequence of states $s_0, s_1, \ldots$, such that $s_0 = s$ and $R(s_i, s_{i+1}) \neq \mathsf{f}$ for every $i \geq 0$. A set of all paths from s is denoted by $\Pi(s)$, and the ith state of a given path $\pi \in \Pi(s)$ – by π_i.

The formal 3-valued semantics of CTL is given below.

$$\begin{array}{lcl}
||\ell||(s) & \triangleq & \ell \\
||p||(s) & \triangleq & I(s,p) \\
||\varphi \wedge \psi||(s) & \triangleq & ||\varphi||(s) \wedge ||\psi||(s) \\
||\neg\varphi||(s) & \triangleq & \neg||\varphi||(s) \\
||EX\varphi||(s) & \triangleq & \bigvee_{s' \in \Sigma}(R(s,s') \wedge ||\varphi||(s')) \\
||EG\varphi||(s) & \triangleq & \bigvee_{\pi \in \Pi(s)} \bigwedge_{i \geq 0} (||\varphi||(\pi_i) \wedge R(\pi_i, \pi_{i+1})) \\
||E[\varphi U \psi]||(s) & \triangleq & \bigvee_{\pi \in \Pi(s)} \bigvee_{i \geq 0} \big(||\psi||(\pi_i) \wedge \\
 & & \quad \bigwedge_{j<i} (R(\pi_j, \pi_{j+1}) \wedge ||\varphi||(\pi_j))\big)
\end{array}$$

In the case of 2-valued state machines, the above semantics of CTL is equivalent to its classical interpretation [4]. The CTL formalization of $\mathbf{P_1}$–$\mathbf{P_5}$ is given in Table 1. For example, $\mathbf{P_1} = AG(\mathsf{c} \Rightarrow EX\mathsf{s})$, i.e., in every state, if focus is achieved, a picture can be taken in one of the next states. Note that refinement preserves valuation of not only positive (e.g., $\mathbf{P_1}, \mathbf{P_2}, \mathbf{P_3}, \mathbf{P_5}$) but also negative (e.g., $\mathbf{P_4}$), universal (e.g. $\mathbf{P_3}$, $\mathbf{P_4}$, $\mathbf{P_5}$,), existential (e.g. $\mathbf{P_2}$), and mixed (e.g. $\mathbf{P_1}$) properties.

We now aim to characterize similarities between models which are not necessarily refinements of each other.

DEFINITION 2. (common refinement) *Let M_1 and M_2 be 3-valued state machines. A 3-valued model M_3 is a* ***common refinement*** *of M_1 and M_2 iff $M_1 \preceq M_3$ and $M_2 \preceq M_3$. Furthermore, M_3 is the* ***least common refinement*** *iff for every common refinement M_4, $M_3 \preceq M_4$.*

Like refinement, common refinement preserves truth and falsity of properties expressed in L_μ [25].

THEOREM 1. *Let M_3 be a common refinement of M_1 and M_2. Then, $\forall \varphi \in L_\mu$:*

$$\begin{array}{lcl}
(||\varphi||^{M_1} = \mathsf{t}) \vee (||\varphi||^{M_2} = \mathsf{t}) & \Rightarrow & ||\varphi||^{M_3} = \mathsf{t} \\
(||\varphi||^{M_1} = \mathsf{f}) \vee (||\varphi||^{M_2} = \mathsf{f}) & \Rightarrow & ||\varphi||^{M_3} = \mathsf{f}
\end{array}$$

Moreover, if M_3 is the least common refinement of M_1 and M_2, then for every common refinement M_4,

$$||\varphi||^{M_4} = \mathsf{m} \Rightarrow ||\varphi||^{M_3} = \mathsf{m}$$

If for some property $\varphi \in L_\mu$, $||\varphi||^{M_1}$ is t and $||\varphi||^{M_2}$ is f, then M_1 and M_2 do not have a common refinement; in this case they are called *inconsistent*.

DEFINITION 3. (consistency) *3-valued models M_1 and M_2 are* ***consistent*** *if their common refinement exists; otherwise, they are* ***inconsistent****.*

To reason about inconsistent models, we introduce a notion of *abstraction*, which is the dual of refinement. *M_1 abstracts M_2* iff M_2 refines M_1. *Common abstraction* of models M_1 and M_2 can be defined in a similar way:

DEFINITION 4. (common abstraction) *Let M_1 and M_2 be 3-valued models. A 3-valued model M_3 is a* ***common abstraction*** *of M_1 and M_2 iff $M_3 \preceq M_1$ and $M_3 \preceq M_2$. Furthermore, M_3 is the* ***greatest common abstraction*** *iff for every common abstraction M_4, $M_4 \preceq M_3$.*

THEOREM 2. *Let M_3 be a common abstraction of M_1 and M_2. Then, $\forall \varphi \in L_\mu$,*

$$\begin{array}{lcl}
||\varphi||^{M_3} = \mathsf{t} & \Rightarrow & (||\varphi||^{M_1} = \mathsf{t}) \wedge (||\varphi||^{M_2} = \mathsf{t}) \\
||\varphi||^{M_3} = \mathsf{f} & \Rightarrow & (||\varphi||^{M_1} = \mathsf{f}) \wedge (||\varphi||^{M_2} = \mathsf{f})
\end{array}$$

Moreover, if M_3 is the greatest common abstraction of M_1 and M_2, then for every common abstraction M_4,

$$||\varphi||^{M_3} = \mathsf{m} \Rightarrow ||\varphi||^{M_4} = \mathsf{m}$$

3.2 Computing Merge

The intuition we wish to capture by merge is that of combining partial knowledge coming from individual models while preserving all of their agreements. The notion of common refinement underlies this intuition as it captures the "more complete than" relation between two incomplete models, and hence we use it in our definition:

DEFINITION 5. (merge) *A **merge** of two 3-valued state-machines is their common refinement.*

Basing the notion of merge on a common refinement is standard [39, 26, 24]. By Theorem 1, the least common refinement preserves most properties of the original models, and thus is the most precise merge; however, it may not necessarily be expressible in 3-valued logic. We discuss this issue further in Section 6.

Clearly, the above definition only applies to consistent models (inconsistent ones simply do not have a common refinement). So, our first goal is to determine whether two models are consistent. We define consistency recursively, similarly to our definition of refinement.

DEFINITION 6. (consistency relation) *Let M_1 and M_2 be 3-valued state machines. We define a **consistency relation** $\sim \subseteq \Sigma_1 \times \Sigma_2$ where $s \sim t$ iff:*

1. $\forall p \in AP_u \cdot I_1(s,p) \sqcup I_2(t,p)$ *is defined*
2. $\forall s' \in \Sigma_1 \cdot R_1(s,s') \succeq \mathsf{t} \Rightarrow \exists t' \in \Sigma_2 \cdot R_2(t,t') \preceq \mathsf{t} \wedge s' \sim t'$
3. $\forall t' \in \Sigma_2 \cdot R_2(t,t') \succeq \mathsf{t} \Rightarrow \exists s' \in \Sigma_1 \cdot R_1(s,s') \preceq \mathsf{t} \wedge s' \sim t'$

We say M_1 and M_2 are consistent, written as $M_1 \sim M_2$, iff $s_0 \sim t_0$.

Intuitively, $s \sim t$ iff values of all propositions in these states are consistent (condition 1), and s and t have consistent successors. The latter means that for every definite, i.e., t, successor s' of s, there exists some possible, i.e., t or m, successor t' of t where s' and t' are consistent (condition 2), and for every definite, i.e., t, successor t' of t, there is some possible, i.e., t or m, successor s' of s, consistent with t' (condition 3). To prove that M_1 and M_2 are consistent, we simply need to match every t transition of one model to some (non-f) transition of the other. m transitions do not need to be matched – they can evolve either to t or to f without causing inconsistency.

THEOREM 3. *M_1 and M_2 have a common refinement iff $M_1 \sim M_2$.*

For example, $\mathbf{CM_1}$ and $\mathbf{CM_2}$ in Figures 1(a)-(b) are consistent with the consistency relation $\{(s_0,t_0),(s_1,t_1),(s_2,t_2),(s_2,t_3)\}$. On the other hand, $\mathbf{CM_2}$ and $\mathbf{CM_3}$ in Figures1(b)-(c) are inconsistent: t_1, the successor of t_0, disagrees with both successors of r_0 (r_1 and r_2) on values of propositions s and c. Since successors of t_0 cannot be matched to successors of r_0, t_0 and r_0 are inconsistent, and thus so are $\mathbf{CM_2}$ and $\mathbf{CM_3}$.

The complexity of computing a relation $\sim$ is equivalent to computing a refinement relation which is polynomial in the size of M_1 and M_2.

If there exists a consistency relation $\sim$ over the states of M_1 and M_2, the construction of a merged model is straightforward: every pair of consistent states is merged to form a single state in the combined model.

DEFINITION 7. ($M_1 + M_2$) *Let M_1 and M_2 be 3-valued state machines, and let $M_1 \sim M_2$. We define a **merge** of M_1 and M_2, denoted $M_1 + M_2$, as a tuple $(\Sigma_1 \times \Sigma_2, (s_0,t_0), R_+, I_+, AP_1 \cup AP_2)$, where for every $(s,t),(s',t') \in \Sigma_1 \times \Sigma_2$,*

$$1. R_+((s,t),(s',t')) = \begin{cases} R_1(s,s') \sqcup R_2(t,t') & \textit{iff } s \sim t \wedge s' \sim t' \\ \mathsf{f} & \textit{otherwise} \end{cases}$$

$$2. \forall p \in AP_u \cdot I_+((s,t),p) = I_1(s,p) \sqcup I_2(t,p)$$

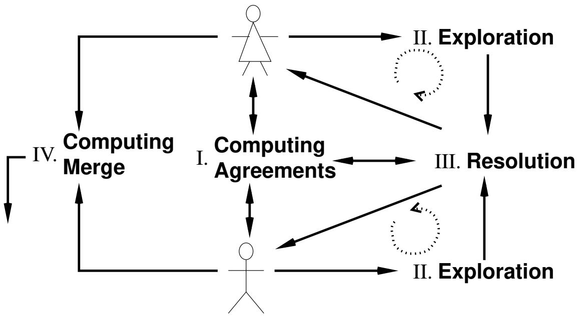

Figure 4: Negotiation framework.

$M_1 + M_2$ is a fragment of the cross-product of M_1 and M_2. Since all transitions between inconsistent pairs of states are f, only consistent pairs are reachable.

THEOREM 4. *Let M_1 and M_2 be 3-valued consistent models. Then, $M_1 + M_2$ is their common refinement.*

For example, model $\mathbf{CM_1} + \mathbf{CM_2}$ is shown in Figure 3(c), where the consistency relation is $\{(s_0,t_0),(s_1,t_1),(s_2,t_2),(s_2,t_3)\}$. For this example, $\mathbf{CM_1} + \mathbf{CM_2}$ is the most precise merge, i.e., the least common refinement, but this is not necessarily the case in general.

4. COPING WITH INCONSISTENCY

When models are inconsistent, their merge does not exist. In this case, we start the negotiation process, outlined in Figure 4. First, we build a model that reflects agreements between the initial models (**Computing Agreements**, Section 4.1). Effectively, we replace all points of potential contention with maybe. Afterwards, we project the result back onto the original models, allowing users to explore the result (**Exploration**, Section 4.2). Using global properties or their intuition, and supported by analysis tools such as model-checkers, users pick a list of items that they care about most. These become input to the **Resolution** phase (Section 4.3) which attempts to handle these disagreements by building consistent proposals, allowing users to pick their favourite. Once proposals have been chosen, the resulting models, which are now consistent but may still be somewhat incomplete, can be merged using techniques described in Section 3.2 (**Computing Merge**). The incompletenesses represent "don't cares" on the part of stakeholders, indicating that they will be satisfied with any consistent resolution of these points.

4.1 Computing Agreements

When models contain inconsistencies, we need to help users identify, understand and resolve them. To this end, it is helpful to construct a *readable* model that preserves the agreements and highlights the disagreements between the models. We refer to it as the *agreement* model and define it as a common abstraction (Definition 4) between the two models. A common abstraction, M_3, of models M_1 and M_2 can distinguish those properties on which M_1 and M_2 agree from those on which they do not: these evaluate to m on M_3. By Theorem 2, the greatest common abstraction, if available, would be a better choice (see Section 6 for a discussion). An even more important problem is that our abstractions cannot distinguish between disagreements that are caused by the lack of information from those caused by the actual disagreement between the models. For example, $\mathbf{P_5}$ evaluates to m in $\mathbf{CM_3}$ because it uses a proposition a, which is not in the context of $\mathbf{CM_3}$; thus, $\mathbf{P_5}$ will be m in every common abstraction involving $\mathbf{CM_3}$, e.g., with

$\mathbf{CM_2}$, even though there is no disagreement. Such problems need not be reported to the user, since they can be resolved simply by refinement. We use a number of heuristics aimed at removing the unnecessary m transitions and variables, such as the one described in our case-study [32]. Further, we project common abstractions back to the context of the original models. Such *projections* can retrieve the information lost by computing the common abstraction. For example, projecting our example common abstraction back to $\mathbf{CM_2}$ ensures that $\mathbf{P_5}$ evaluates to t. We describe the construction of agreement and projection models below.

DEFINITION 8. ($M_1 \oplus M_2$) *Let M_1 and M_2 be 3-valued state machines. The **agreement** of M_1 and M_2, denoted $M_1 \oplus M_2$, is a tuple $(\Sigma_1 \times \Sigma_2, (s_0, t_0), R_\oplus, I_\oplus, AP_1 \cup AP_2)$ where for every $(s,t), (s', t') \in \Sigma_1 \times \Sigma_2$,*

$$\begin{array}{lllll} 1. & R_\oplus((s,t),(s',t')) & = & R_1(s,s') \sqcap R_2(t,t') \\ 2. & \forall p \in AP_u \cdot I_\oplus((s,t),p) & = & I_1(s,p) \sqcap I_2(t,p) \end{array}$$

$M_1 \oplus M_2$ is the "dual" of $M_1 + M_2$: every join ($\sqcup$) in the latter is replaced with a meet ($\sqcap$) in the former.

THEOREM 5. *Given 3-valued models M_1 and M_2, $M_1 \oplus M_2$ is their common abstraction.*

Theorem 5 gives us a procedure for computing agreements and disagreements between M_1 and M_2. However, $M_1 \oplus M_2$ includes the entire cross-product of states of M_1 and M_2! $M_1 + M_2$ was relatively small because its state-space was limited to consistent pairs of states of M_1 and M_2, but the state-space of $M_1 \oplus M_2$ is $\Sigma_1 \times \Sigma_2$. Even if it is computationally feasible (afterall, we expect that the models are relatively small), models with such number of states are difficult for analysts to understand, since they differ drastically from their original models. Instead, we seek a definition of maximum agreement that remains a common abstraction of M_1 and M_2 while being more readable.

Our goal is to reduce the state-space of the agreement model from $\Sigma_1 \times \Sigma_2$ to a set $\rho \subseteq \Sigma_1 \times \Sigma_2$ representing maximal agreements between M_1 and M_2. We need ρ to be left/right total:

$$(\forall s \in \Sigma_1 \cdot \exists t \in \Sigma_2 \cdot (s,t) \in \rho) \wedge (\forall t \in \Sigma_2 \cdot \exists s \in \Sigma_1 \cdot (s,t) \in \rho)$$

and further, $(s_0, t_0) \in \rho$. We restrict the state-space of $M_1 \oplus M_2$ to ρ by redefining $R_\oplus$ as follows:

$$R_\oplus((s,t),(s',t')) = \begin{cases} R_1(s,s') \sqcap R_2(t,t') & \textit{iff } (s,t),(s',t') \in \rho \\ \mathsf{f} & \textit{otherwise} \end{cases}$$

Note that restricting the state-space of $M_1 \oplus M_2$ to a ρ with above-mentioned properties still yields a common abstraction of M_1 and M_2.

THEOREM 6. *Let ρ be a left/right total relation s.t. $(s_0, t_0) \in \rho$. Then, $M_1 \oplus M_2$ built using ρ is a common abstraction of M_1 and M_2.*

Different ρs result in different agreement models. Naturally, we are interested in those ρs that maximize agreements. By definition of $I_\oplus$ and $R_\oplus$, a maybe variable or transition in $M_1 \oplus M_2$ may indicate a disagreement between M_1 and M_2. Thus, we want to obtain a relation ρ that induces a minimal number of such entities in $M_1 \oplus M_2$. For example, the agreement model $\mathbf{CM_2} \oplus \mathbf{CM_3}$, corresponding to the camera models $\mathbf{CM_2}$ and $\mathbf{CM_3}$ in Figures 1(b)-(c), is shown in Figure 3(d). It is built using the relation $\{(t_0, r_0), (t_1, r_1), (t_2, r_2), (t_3, r_2)\}$, which represents maximum agreement over the states of $\mathbf{CM_2}$ and $\mathbf{CM_3}$. After mapping the initial states t_0 and r_0 to each other, we have the option of mapping t_1 to either (or both) of r_1 or r_2. However, r_1 and t_1 only disagree on s, whereas t_1 and r_2 disagree on both s and c. Since t_1 and r_1 are "more similar", they appear in $\mathbf{CM_2} \oplus \mathbf{CM_3}$. For the same reason, we map t_2 and t_3 to r_2 instead of r_0.

Finding the best ρ is essentially an optimization problem, with complexity exponential in the number of the states of M_1 and M_2. While this may still be feasible for relatively small models, we can use various heuristics instead. One of these is described in Section 4.3. Further investigation into effective computations of the best ρ is left for future work.

We now move to the subject of computing projections of common abstractions of state-machine models.

DEFINITION 9. ($(M_1 \oplus M_2)/M_1$) *A **projection** of $M_1 \oplus M_2$ onto M_1, denoted by $(M_1 \oplus M_2)/M_1$, is a tuple $(\Sigma_1, s_0, R_{\oplus/1}, I_{\oplus/1}, AP_1)$, where for every $s, s' \in \Sigma_1$,*

$$1. R_{\oplus/1}(s,s') = \bigsqcap_{(s,t),(s',t') \in \rho} R_\oplus((s,t),(s',t'))$$

$$2. I_{\oplus/1}(s,p) = \begin{cases} \bigsqcap_{(s,t)\in\rho} I_\oplus((s,t),p) & \textit{if } p \in AP_1 \cap AP_2 \\ I_1(s,p) & \textit{otherwise} \end{cases}$$

For conciseness, we use $\hat{M}_1$ to refer to $(M_1 \oplus M_2)/M_1$. The statespace and the context of $\hat{M}_1$ are the same as those of M_1. Transitions and propositions in $\hat{M}_1$ are induced by those in $M_1 \oplus M_2$. However, values of propositions which are not present in M_2 (and thus are m in $M_1 \oplus M_2$), are determined by M_1. $\hat{M}_2$ is defined similarly.

$\mathbf{E}$ and $\mathbf{F}$, shown in Figures 3(e)-(f), are projections corresponding to $\mathbf{CM_2}$ and $\mathbf{CM_3}$, respectively. Since variable a is not in the context of $\mathbf{CM_3}$, its value in each state of $\mathbf{E}$ is set to that in the corresponding state of $\mathbf{CM_2}$. Recall that $\mathbf{P_5}$ was inconclusive on $\mathbf{CM_2} \oplus \mathbf{CM_3}$, but it holds in $\mathbf{E}$, so no further negotiation w.r.t. this property is required.

Projection models, while abstract the corresponding original models, are mutually consistent.

THEOREM 7. *$\hat{M}_1$ and $\hat{M}_2$ are consistent. Further, $\hat{M}_1$ and M_2 as well as M_1 and $\hat{M}_2$ are pair-wise consistent.*

Note that ρ, used in computing $M_1 \oplus M_2$, is a consistency relation between $\hat{M}_1$ and $\hat{M}_2$ as well as between $\hat{M}_1$ and M_2, and between M_1 and $\hat{M}_2$.

4.2 Exploration

The **Computing Agreements** phase produces consistent but incomplete projections $\hat{M}_1$ and $\hat{M}_2$. The missing information, represented by m in the two models, effectively comes from "backing down" from all disagreements between the original models. Clearly, $\hat{M}_1$ and $\hat{M}_2$ can be merged, but the result leaves a number of properties, perhaps the ones which are vitally important to the stakeholders, inconclusive. On the other extreme, we can attempt to reach agreement over every m item (i.e., a variable or a transition). However, as we show in Section 4.3, the number of proposals for resolving inconsistencies can grow exponentially with the number of items; thus, the smaller the list of negotiation items, the easier it is for the stakeholders to reach agreement.

The goal of the (optional) **Exploration** phase is to choose the truly important items, which must be negotiated, from the overall list. We call this a *priority list (PL)*. Leaving an item off the PL indicates that the stakeholder is content with it becoming *either* t *or* f at some point in the future. Note also that each stakeholder builds her own PL independently, so if an item is really important to one stakeholder and not important to the other, the resolution is to simply choose the second user's value for this item.

In order to build the PLs, users can either informally inspect the projections or use various analysis tools such as model-checkers, simulators, debuggers etc. Specifically, if a set of global properties is available, users may want to see the impact of rolling back the disagreements on these. In this paper, we assumed that global properties are expressed in temporal logic, so any 3-valued model-checker, e.g., χChek [5], can be used for this analysis. Stakeholders may further prioritize those properties on which the analysis ended up being inconclusive, and restrict their PL just to those items that caused inconclusiveness of these most desirable properties. This information is also readily obtainable from a model-checking run. Specifically, χChek can return a counterexample explaining why the property evaluates to m. It also has a feature which returns *all* reasons why a property is m, in the form of an *abstract counterexample* [6]. For our framework, we modified χChek to extract the list of m variables and transitions from the returned counterexample and report its size to the user. This gives her an early indication about the feasibility of achieving agreements during the resolution step.

For example, suppose the owner of $\mathbf{CM_2}$ is most interested in the property $\mathbf{P_2}$. A model-checker reports that $\mathbf{P_2}$ is m on the projection $\mathbf{E}$ (see Figure 3(e)) because c is only true in state t_1 and $\neg$s is only true in state t_0, but t_0 is an m-successor of t_1. Thus, the cause of inconclusiveness of $\mathbf{P_2}$ is the transition from t_1 to t_0, which reduces the list of negotiation items for this user from four (three m transitions, one m variable) to one. Similarly, the owner of $\mathbf{CM_3}$ can identify that the m value of s in state r_1 in his projection shown in Figure 3(f) is the cause of inconclusiveness of his most desirable property $\mathbf{P_3}$.

4.3 Resolution

The goal of the **Resolution** phase is to compute alternatives for resolving the most important inconsistencies identified during the **Exploration** phase, while making minimal possible modifications to the original models. The algorithm receives as input the projection models $\hat{M}_1$ and $\hat{M}_2$, a consistency relation ρ between $\hat{M}_1$ and $\hat{M}_2$ (see Theorem 7), and the priority lists of inconsistencies, obtained by merging the individual PLs. It either computes a list of model pairs $(\hat{M_1}', \hat{M_2}')$, called *proposals*, which resolve these inconsistencies, or reports a failure. The goal of the algorithm is to change the value of every m proposition or transition in the combined priority list to either t or f, resulting in models $\hat{M_1}'$ and $\hat{M_2}'$ which remain consistent with respect to ρ.

The steps of the resolution algorithm are shown in Figure 5. Figures 5(a)-(c) show the resolution of m propositions. Suppose we want to resolve a value of a proposition p in state s of $\hat{M}_1$. Let $T = \{t_1, \ldots, t_n\} \subseteq \Sigma_2$ be a set of states mapped by ρ to s, i.e., $\forall t_i \in T \cdot (s, t_i) \in \rho$. We distinguish three possibilities:

Case 1. All states in T are consistent with each other on the value of p, and there are some states $t_i \in T$ where p is t (or f) (see Figure 5(a)). In this case, the value of p in s is set to that of p in t_i, and one proposal $(\hat{M_1}', \hat{M_2}')$ is generated. Note that $\hat{M_2}'$ remains equal to $\hat{M}_2$: we are able to resolve inconsistencies, and additional changes to $\hat{M}_2$ are not needed.

Case 2. All states in T are consistent with each other on the value of p, and no state $t_i \in T$ is conclusive for p (see Figure 5(b)). In this case, the value of p in s is set once to t and once to f. The resulting proposals $(\hat{M_1}', \hat{M_2}')$ and $(\hat{M_1}'', \hat{M_2}'')$ are shown in Figure 5(b). This corresponds to resolving a *point inconsistency*.

Case 3. States in T are inconsistent on p (see Figure 5(c)). In this case, a conflict is reported because changing the value of p in s to either t or f violates the consistency relation ρ. Tuples that cause conflicts, e.g., (s, t) and (s, t') in Figure 5(c), are reported back to the **Computing Agreements** phase (see Figure 4), which attempts to find a better ρ, and then the resolution process is repeated. In principle, we can start with a ρ that maps all states of the original models to each other, and iteratively refine it by finding (during **Resolution**) and removing (during **Computing Agreements**) tuples that cause disagreement.

The resolution of m transitions proceeds similarly, and the algorithm is shown in Figures5(d)-(f). Here, a transition from s to s' is mapped to the one from t to t' iff both tuples $(s, t), (s', t')$ are in ρ. We obtain all possible proposals by resolving every element in the joint PL.

For example, suppose the resolution algorithm is given models $\mathbf{E}$ and $\mathbf{F}$ (see Figures 3(e)-(f)) and the following priority lists: $t_1 \rightarrow t_0$ and s in t_1 in model $\mathbf{E}$, and s in r_1 and $r_0 \rightarrow r_2$ in model $\mathbf{F}$. Recall that the consistency relation ρ for $\mathbf{E}$ and $\mathbf{F}$ was $\{(t_0, r_0), (t_1, r_1), (t_2, r_2), (t_3, r_2)\}$. The algorithm yields eight resolution proposals, one of which, $(\mathbf{G}, \mathbf{H})$, is shown in Figures 3(g)-(h). In $\mathbf{G}$, the value of $t_1 \rightarrow t_0$ is set to t, and s in t_1 is set to f. In $\mathbf{H}$, s in r_1 is set to f, and thus there is no inconsistency between r_1 and t_1. Also, $r_0 \rightarrow r_2$ is set to f.

The complexity of resolving each individual item depends on the number of states or transitions affected by it, which is a (small) fraction of ρ. However, the number of generated proposals is exponential in the size of the joint list because the resolution of each item can potentially lead to two proposals. Fortunately, items in the priority list often depend on each other, effectively reducing the overall number of generated proposals. In our running example, resolution yielded just eight proposals for four priority items: since ρ maps t_1 to r_1, the value of s in t_1 has to be the same as in r_1, reducing the number of non-redundant elements in the joint PL to three. Still the number of proposals may be large. We envision that in such situations, users will partition their PLs, so that the negotiations can concentrate only on the chosen items. Additional iterations of the framework would be required to resolve the remaining items. Effectively, this enables compositional negotiation. We are still working on a methodology to help users partition their PLs.

Proposals generated by the resolution algorithm are still consistent with respect to ρ, just like the corresponding projections have been, but they refine these projections, deeming more properties conclusive. These two facts establish correctness of the algorithm, and are summarized in the theorem below.

THEOREM 8. *For every proposal $(\hat{M_1}', \hat{M_2}')$ generated by the algorithm in Figure 5, $\hat{M_1}'$ and $\hat{M_2}'$ are consistent with respect to ρ, and further, $\hat{M}_1 \preceq \hat{M_1}'$ and $\hat{M}_2 \preceq \hat{M_2}'$.*

On the other hand, the relationship between proposals and the *original* models M_1 and M_2 is *orthogonal* to refinement: we first abstract from all inconsistencies, yielding maybes, and then refine the result consistently. For example, if a property φ was t in M_1 and f in M_2, it becomes m in $\hat{M}_1$ and $\hat{M}_2$, and, if present in the PL, is set either to t or to f in *both* $\hat{M_1}'$ and $\hat{M_2}'$. Thus, inconsistency resolution is *non-monotonic*!

As proposals are being generated, users can explore them and, if satisfied, accept one. The resulting models are guaranteed to give conclusive values to items on the priority list and be consistent with respect to ρ; thus, they can be easily merged. This is done in the **Computing Merge** phase (see Figure 4) and computed using the techniques described in Section 3.2. For example, assuming that the users choose the proposal $(\mathbf{G}, \mathbf{H})$, the resulting merge, $\mathbf{I}$, is shown in Figure 3(i). Note that this example illustrates the difference between defining a merge as refinement versus as a union of transition relations of the original models. For example, the m-

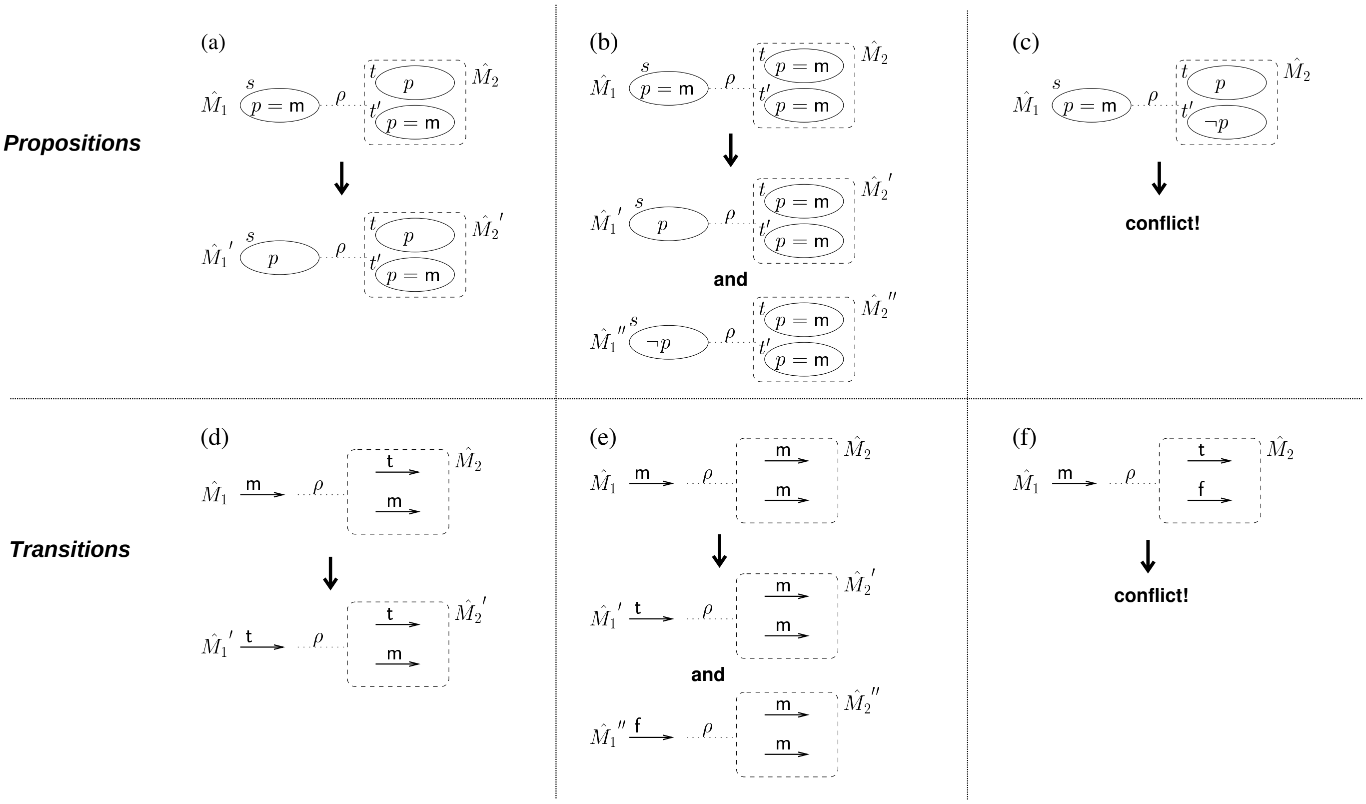

Figure 5: The steps of the resolution algorithm: (a-c) the resolution of maybe propositions; (d-f) the resolution of maybe transitions.

transitions from t_0 to t_2 and t_3 in $\mathbf{G}$ are not present in the merged model $\mathbf{I}$. Thus, the transitions of $\mathbf{I}$ are not the union of the transitions of $\mathbf{G}$ and $\mathbf{H}$.

Two other cases can occur: (a) no proposals can be generated; and (b) users do not consider any of the proposals to be acceptable. We discussed the former earlier, under *Case 3* of the resolution algorithm. In the latter case, the stakeholders can go back to the exploration phase and produce a different PL, or perhaps decide that the problem is with the existing ρ and give hints on how to modify it. Currently this process is manual (the **Computing Agreements** phase can optionally read in a suggested ρ), but we are looking for ways of automating it.

5. TOOL SUPPORT AND PRELIMINARY EVALUATION

We have prototyped a proof of concept implementation of the merge and negotiation framework discussed in earlier sections. The implementation can merge 3-valued state-machines if they are consistent (Definitions 6 and 7). If models are inconsistent, it computes an agreement model, and, using a multi-valued model-checker χChek [5], allows stakeholders to explore its projections, building a priority list from χChek's abstract counterexamples. Once proposals are built (using the algorithm in Figure 5), users can choose their favourite, apply suggested resolutions to their *original* models, and attempt to merge them again. This allows *incremental* negotiation. Our implementation supports it by additionally storing the history of made decisions, allowing users to go back and undo them (and thus facilitating "what if" exploration).

To try our framework on a more realistic example, we attempted to merge inconsistent descriptions of behaviour of an authentication system, adopted from [38]. The system is described from the administrator's and from the user's points of view, and the models disagree on a property "Entering a password can be followed by a successful authentication". The models were translated from MTSs [31] to 3-valued state machines and had 3 and 5 states, respectively, with the combined vocabulary of 3 variables. The size of ρ was 5, i.e., $max(\Sigma_1, \Sigma_2)$. We computed the maximal agreement model and automatically refined it using a heuristic discussed in [32]. Using the above-mentioned property, we identified only two priority items, which ended up being related. The resolution algorithm yielded two proposals, forcing the property into becoming t or f in both models, respectively. We note that one of the resulting proposals is exactly the modification of the user model done by hand in [38] in order to resolve inconsistencies, and that ρ is the same as the consistency relation obtained in [38]. Further details are available in [32].

6. IMPROVING PRECISION

In Sections 3.2 and 4.1, we defined the least common refinement and the greatest common abstraction as the most precise merge and the most precise agreement model, respectively. However, we noted that we could not necessarily construct these. For example, models $\mathbf{J}$ and $\mathbf{K}$ in Figures 6(a)-(b) are consistent, but their 3-valued merge, $\mathbf{L}$, shown in Figure 6(c), is clearly not the most precise: neither of the original systems specified whether there is a successor to the initial state where both p and q hold, i.e., a CTL property $EX(p \wedge q)$ evaluates to m on both $\mathbf{J}$ and $\mathbf{K}$. This means that this property can possibly become t or f in future refinements of both models. However, $\mathbf{L}$ disallows this possibility: $EX(p \wedge q)$ simply evaluates to t.

We can capture the most precise merge with 4-valued logic [1], using the additional value, d (see Figure 7 for the information, $\preceq$,

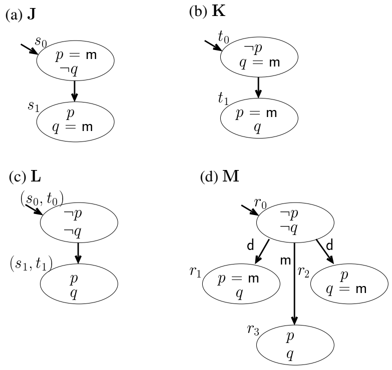

Figure 6: (a) and (b) two simple models; (c) their 3-valued merge; and (d) their most precise, 4-valued merge.

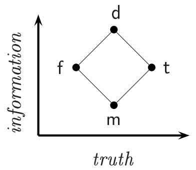

Figure 7: 4-valued logic.

and the truth, $\leq$, orderings for this logic). This logic has been originally proposed to handle inconsistency [1], but in our work, it can be used to improve precision. For example, the 4-valued common refinement, $\mathbf{M}$, of models $\mathbf{J}$ and $\mathbf{K}$, shown in Figure 6(d), is consistent. d and m are used to identify transitions between states which are related to one another. For example, state r_3 is a refinement of states r_1 and r_2; d transitions from r_0 (the unified initial state) to r_1 and r_2, combined with an m transition to r_3, precisely express the least common refinement and guarantee that $EX(p \wedge q)$ is m.

In general, we extend our 3-valued state-machines in a natural way to allow 4-valued transitions. We never need to capture 4-valued propositions because these may result in inconsistent models– a topic which is outside the scope of this paper. For more information, the reader can refer to [19]. The definitions of refinement and abstraction over 4-valued models are exactly those given in Definitions 1 and 4; moreover, existence and uniqueness of the least common refinement and the greatest common abstraction are guaranteed in the world of 4-valued models [33]. Since our underlying model-checker χChek can handle Belnap logic as well, we can easily obtain a framework that facilitates creation of most precise merges.

Consider the models $\mathbf{L}$ and $\mathbf{M}$ again. Even though the latter is more precise, the former is more concise and readable, and the difference between the 3- and the 4-valued merges becomes drastic as the size of the original models increases. Our preliminary experience showed that the resulting models are difficult for the stakeholders to understand, undermining the high interactivity of the proposed framework. Thus, so far we concentrated on readability on lieu of precision, building 3-valued merges and common agreement models, as described earlier in this paper. However, we still believe that a compromise between readability and precision may be achieved, leaving further exploration of this point for future work.

7. RELATED WORK

In this paper, we addressed model merging and techniques for exploration, resolution and negotiation of inconsistencies. Effectively, resolution is a form of model evolution, which stems from taking the other point of view into account.

Our definition of merge is similar to the work of Uchitel and Chechik [39]. Like ours, their notion of merge is based on common refinement, relaxed in [39] to *observational refinement*. They consider a version of modal transition systems (MTSs [25]) which are closely related to our 3-valued state machines. However, while able to detect inconsistencies, [39] does not consider the problem of negotiation and conflict resolution.

A number of approaches to inconsistency management have been studied in the context of viewpoint-based modeling [35]. Some of this work, e.g., [14, 34], detects inconsistencies by using first-order logic rules and does not consider merge as a means of model exploration and inconsistency detection. Other researchers [26, 13, 37] propose ways of merging viewpoint models. Huth and Pradhan [26] define the merge as the common refinement of partial state transition systems. They enforce consistency across inconsistent viewpoints by using a dominance ordering on owners of the viewpoints. [13, 37] allow the merge of *inconsistent* viewpoints using multi-valued logic, like we do in this paper. The goal of this work is to tolerate disagreement while still enabling reasoning. In [13], states are merged, i.e., put into ρ, only if they have the same label. The merge in [37] is based on the structural mapping of graph morphisms with the emphasis on preserving structure rather than behavior. Non-classical logics have been used for reasoning with inconsistency by others, e.g., in [23]. Unlike this work, our goal is to merge only consistent models, and we allow users to explore agreement models, which are consistent but incomplete, to determine those inconsistencies that need to be resolved.

In [11], requirements evolution is supported by an iterative process that is similar to our exploration and resolution steps; however, the implementation of the resolution phase is left open, conjecturing that machine learning techniques may be suitable for it. The work in [15] extends [11] by using multi-valued logic to address incompleteness and partiality of requirements specifications. However, [15] does not give the problem a formal treatment, illustrating the process by an example instead.

We are not aware of other formal logic-based approaches to negotiation. The existing work, e.g., [9, 12, 2], is based on dialectic reasoning. In these approaches, requirements are treated as informal generic entities, and the focus is on formalizing the relationships and interactions between them. Even though both functional and non-functional requirements can be handled using these approaches, correctness and completeness become subjective. In contrast, we only consider behavioural requirements, but their formality allowed us to perform tool-supported inconsistency resolution while tolerating the less critical inconsistencies.

8. CONCLUSION AND FUTURE WORK

In this paper, we have described a formal framework for merge and conflict resolution. This framework facilitates automatic merging of consistent models, enables users to visualize and explore potential disagreements and identify their priorities, and suggests ways to resolve the priority items.

Several research problems need to be solved to ensure that this framework is effective. The first and most important of these is the efficient computation of a relation ρ that reduces the size of the agreement models while capturing the maximal similarities between the inconsistent models. Computing an optimal ρ is similar to the *schema matching* problem – a subject that has been extensively studied in the database literature, e.g., [36]. We are currently looking for ways to tailor the existing schema matching techniques to our framework. We further need to evaluate the effectiveness of the framework on more realistic case-studies as well as develop additional heuristics to improve precision of the merge and a methodology for partitioning PLs in cases when the framework generates too many proposals.

Another direction is changing the resolution algorithm to produce more proposals. Our algorithm produces proposals obtained by keeping the relation ρ intact. More interesting proposals can be produced if we allow the algorithm to extend ρ. In this paper, we studied negotiation over flat state-machines. Adding hierarchy as well as more complex language features would enable merging and negotiation over more realistic models.

9. ACKNOWLEDGMENTS

We thank Mehrdad Sabetzadeh and Sebastian Uchitel for providing us with the examples and for their helpful comments. This research was partially funded by NSERC and MITACS.

10. REFERENCES

[1] N. Belnap. “A Useful Four-Valued Logic”. In *Modern Uses of Multiple-Valued Logic*. Reidel, 1977.

[2] B. Boehm and H. In. “Identifying Quality-Requirement Conflicts”. *IEEE Software*, 13(2), 1996.

[3] G. Bruns and P. Godefroid. “Generalized Model Checking: Reasoning about Partial State Spaces”. In *CONCUR*, 2000.

[4] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. “Multi-Valued Symbolic Model-Checking”. *ACM TOSEM*, 12(4), 2003.

[5] M. Chechik, B. Devereux, and A. Gurfinkel. “χChek: A Multi-Valued Model-Checker”. In *CAV*, 2002.

[6] M. Chechik and A. Gurfinkel. “A Framework for Counterexample Generation and Exploration”. In *FASE*, 2005.

[7] S. Clarke and R. J. Walker. “Composition Patterns: An Approach to Designing Reusable Aspects”. In *ICSE*, 2001.

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

[9] J. Conklin and M. Begeman. “gIBIS: A Hypertext Tool for Exploratory Policy Discussion”. *Trans. on Info. Sys.*, 4(6), 1988.

[10] D. Damian, A. Eberlein, M. Shaw, and B. Gaines. “An Exploratory Study of Facilitation in Distributed Requirements Engineering”. *REJ*, 8(1), 2003.

[11] A. d’Avila Garcez, A. Russo, B. Nuseibeh, and J. Kramer. “An Analysis-Revision Cycle to Evolve Requirements Specifications”. In *ASE*, 2001.

[12] S. Easterbrook. “Resolving Conflicts Between Domain Descriptions with Computer-Supported Negotiation”. In *Workshop on Knowledge Acquisition for Knowledge Based Systems*, 1990.

[13] S. Easterbrook and M. Chechik. “A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints”. In *ICSE*, 2001.

[14] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency Handling in Multi-Perspective Specifications”. *IEEE TSE*, 20(8), 1994.

[15] J. Garcia-Duque, J. Pazos-Arias, and B. Barragans-Martinez. “An Analysis-Revision Cycle to Evolve Requirements Specifications by Using the SCTL-MUS Methodology”. In *ICRE*, 2002.

[16] P. Godefroid and R. Jagadeesan. “On the Expressiveness of 3-Valued Models”. In *VMCAI*, 2003.

[17] T. Gruber. “The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases.”. In *Principles of Knowledge Representation and Reasoning*, 1991.

[18] A. Gurfinkel and M. Chechik. “Multi-Valued Model-Checking via Classical Model-Checking”. In *CONCUR*, 2003.

[19] A. Gurfinkel and M. Chechik. “Yasm: Model-Checking Software with Belnap Logic”, 2005. Univ. of Toronto Tech. Report.

[20] D. Harel. “StateCharts: A Visual Formalism for Complex Systems”. *Science of Computer Programming*, 8, 1987.

[21] M. Heimdahl and N. Leveson. “Completeness and Consistency in Hierarchical State-Based Requirements”. *IEEE TSE*, 22(6), 1996.

[22] C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw. “SCR*: A Toolset for Specifying and Analyzing Requirements”. In *COMPASS*, 1995.

[23] A. Hunter and B. Nuseibeh. “Managing Inconsistent Specifications: Reasoning, Analysis and Action”. *ACM TOSEM*, 7(4), 1998.

[24] A. Hussain and M. Huth. “On Model Checking Multiple Hybrid Views”. In *Inter. Symp. on Leveraging Apps. of FMs*, 2004.

[25] M. Huth, R. Jagadeesan, and D. A. Schmidt. “Modal Transition Systems: A Foundation for Three-Valued Program Analysis”. In *ESOP*, 2001.

[26] M. Huth and S. Pradhan. “Model-Checking View-Based Partial Specifications”. *Electr. Notes Theor. Comp. Sci.*, 45, 2001.

[27] D. Jackson. “Alloy: A Lightweight Object Modelling Notation”. *ACM TOSEM*, 11(2), 2002.

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

[29] D. Kozen. “Results on the Propositional μ-calculus”. *TCS*, 27, 1983.

[30] I. Krueger, R. Grosu, P. Scholz, and M. Broy. “From MSCs to Statecharts”. In *Conf. Distributed and Parallel Embedded Systems*, 1999.

[31] K. Larsen and B. Thomsen. “A Modal Process Logic”. In *LICS*, 1988.

[32] S. Nejati. “Negotiation for a B2B E-Commerce Site”. `http://www.cs.toronto.edu/~shiva/examples`.

[33] S. Nejati. “Abstraction and Software Model Checking”, 2005. Univ. of Toronto Depth Paper.

[34] C. Nentwich, W. Emmerich, A. Finkelstein, and E. Ellmer. “Flexible Consistency Checking”. *ACM TOSEM*, 12(1), 2003.

[35] B. Nuseibeh, J. Kramer, and A. Finkelstein. “Framework for Expressing the Relationship Between Multiple Views in Requirements Specifications”. *IEEE TSE*, 20(10), 1994.

[36] E. Rahm and P. A. Bernstein. “A Survey of Approaches to Automatic Schema Matching”. *VLDB Journal*, 10(4), 2001.

[37] M. Sabetzadeh and S. Easterbrook. “Analysis of Inconsistency in Graph-Based Viewpoints”. In *ASE*, 2003.

[38] S. Uchitel and M. Chechik. “Merging MTSs for a B2B E-Commerce Site”. `http://www.doc.ic.ac.uk/~su2/merge/examples`.

[39] S. Uchitel and M. Chechik. “Merging Partial Behavioural Models”. In *FSE*, 2004.

[40] S. Uchitel, J. Kramer, and J. Magee. “Synthesis of Behavioural Models from Scenarios”. *IEEE TSE*, 29(2), 2003.

APPENDIX

A. SELECTED THEOREMS

In this Appendix, we give proof sketches for some of the theorems that appeared in the paper. Theorems 1 and 2 are from [25]. The proofs of Theorems 7 and 8 follow from the construction of projections and the resolution algorithm, respectively.

Theorem 3. M_1 and M_2 have a common refinement iff $M_1 \sim M_2$.

Proof:

$\Rightarrow$ Let M_3 be a common refinement of M_1 and M_2. Then, there are two refinement relations $\preceq$ and $\preceq'$ s.t. $M_1 \preceq M_3$ and $M_2 \preceq' M_3$. We define a relation $\rho \subseteq \Sigma_1 \times \Sigma_2$ as follows:

$$\rho = \{(s,t) \in \Sigma_1 \times \Sigma_2 \mid \exists r \in \Sigma_3 \cdot s \preceq r \wedge t \preceq' r\}$$

Informally, ρ contains tuples (s,t) where s and t have a common refinement r. It can be proven that ρ is a consistency relation between M_1 and M_2, i.e., ρ satisfies the conditions in Definition 6. Thus, $M_1 \sim M_2$.

$\Leftarrow$ Let $M_1 \sim M_2$. Then, by Theorem 4, $M_1 + M_2$ is a common refinement of M_1 and M_2.

□

Theorem 4. Let M_1 and M_2 be 3-valued consistent models. Then, $M_1 + M_2$ is their common refinement.

Before we give the proof, we provide an inductive definition, equivalent to Definition 1, for the refinement relation $\preceq$.

DEFINITION 10. *We define a sequence of refinement relations $\preceq^0, \preceq^1, \ldots$ on $\Sigma_1 \times \Sigma_2$ as follows:*
- *$s \preceq^0 t$ iff $I_1(s,p) \preceq I_2(t,p)$ for all $p \in AP_u$, and*
- *$s \preceq^{n+1} t$ iff*
  1. $\forall p \in AP_u \cdot I_1(s,p) \preceq I_2(t,p)$
  2. $\forall s' \in \Sigma_1 \cdot R_1(s,s') \succeq \mathsf{t} \Rightarrow \exists t' \in \Sigma_2 \cdot R_2(t,t') \succeq \mathsf{t} \wedge s' \preceq^n t'$
  3. $\forall t' \in \Sigma_2 \cdot R_2(t,t') \preceq \mathsf{t} \Rightarrow \exists s' \in \Sigma_1 \cdot R_1(s,s') \preceq \mathsf{t} \wedge s' \preceq^n t'$

We say $s \preceq t$ iff $s \preceq^i t$, for all $i \geq 0$.

Note that since M_1 and M_2 are finite structures, the sequence $\preceq^0, \preceq^1, \ldots$ is finite as well.

Proof:
We proceed in two steps:

I. We first show that for every $s \in \Sigma_1$ if $s \sim t$, then $s \preceq (s,t)$. It suffices to show $s \sim t \Rightarrow s \preceq^i (s,t)$ for all $i \geq 0$. We prove it by induction on i:

Base case. $s \sim t \Rightarrow s \preceq^0 (s,t)$.

$s \preceq^0 (s,t)$
$\Leftrightarrow$ (by the definition of $\preceq^0$)
$\forall p \in AP_u \cdot I_1(s,p) \preceq I_+((s,t),p)$
$\Leftrightarrow$ (since $I_+((s,t),p) = I_1(s,p) \sqcup I_2(t,p)$)
$\forall p \in AP_u \cdot I_1(s,p) \preceq I_1(s,p) \sqcup I_2(t,p))$
$\Leftrightarrow$ (by the properties of $\sqcup$)
true

Inductive case. Suppose $s \sim t \Rightarrow s \preceq^n (s,t)$. We prove that

$$s \sim t \Rightarrow s \preceq^{n+1} (s,t)$$

By Definition 10, we need to show:

1. $I_1(s,p) \preceq I_+((s,t),p)$
2. $\forall s' \in \Sigma_1 \cdot R_1(s,s') \succeq \mathsf{t} \Rightarrow \exists (s',t') \in \Sigma_1 \times \Sigma_2 \cdot$
$R_+((s,t),(s',t')) \succeq \mathsf{t} \wedge s' \preceq^n (s',t')$
3. $\forall (s',t') \in \Sigma_1 \times \Sigma_2 \cdot R_+((s,t),(s',t')) \preceq \mathsf{t} \Rightarrow$
$\exists s' \in \Sigma_1 \cdot R_1(s,s') \preceq \mathsf{t} \wedge s' \preceq^n (s',t')$

1. From $I_+((s,t),p) = I_1(s,p) \sqcup I_2(t,p)$, and $I_1(s,p) \preceq I_1(s,p) \sqcup I_2(t,p)$

2.

$\forall s' \in \Sigma_1 \cdot R_1(s,s') \succeq \mathsf{t}$
$\Rightarrow$ (since $s \sim t$ and by Definition 6, condition 2)
$R_1(s,s') \succeq \mathsf{t} \wedge \exists t' \in \Sigma_2 \cdot R_2(t,t') \preceq \mathsf{t} \wedge s' \sim t'$
$\Rightarrow$ (by the properties of $\sqcup$)
$\exists t' \in \Sigma_2 \cdot R_1(s,s') \sqcup R_2(t,t') \succeq \mathsf{t} \wedge s' \sim t'$
$\Rightarrow$ (since $R_+((s,t),(s',t')) = R_1(s,s') \sqcup R_2(t,t')$)
$\exists (s',t') \in \Sigma_1 \times \Sigma_2 \cdot R_+((s,t),(s',t')) \succeq \mathsf{t} \wedge s' \sim t'$
$\Rightarrow$ (by the inductive hypothesis)
$\exists (s',t') \in \Sigma_1 \times \Sigma_2 \cdot R_+((s,t),(s',t')) \succeq \mathsf{t} \wedge s' \preceq^n (s',t')$

3.

$\forall (s',t') \in \Sigma_1 \times \Sigma_2 \cdot R_+((s,t),(s',t')) \preceq \mathsf{t}$
$\Rightarrow$ (by the definition of R_+)
$R_1(s,s') \sqcup R_2(t,t') \preceq \mathsf{t} \wedge s' \sim t'$
$\Rightarrow$ (by $\sqcup$ properties)
$\exists s' \in \Sigma_1 \cdot R_1(s,s') \preceq \mathsf{t} \wedge s' \sim t'$
$\Rightarrow$ (by the inductive hypothesis)
$\exists s' \in \Sigma_1 \cdot R_1(s,s') \preceq \mathsf{t} \wedge s' \preceq^n (s',t')$

II. Similarly, we show that for every $t \in \Sigma_2$, if $s \sim t$, then $t \preceq (s,t)$.

Since M_1 and M_2 are consistent, we have $s_0 \sim t_0$. By **I.** and **II.**, we obtain $s_0 \preceq (s_0,t_0)$ and $t_0 \preceq (s_0,t_0)$. This implies that $M_1 \preceq M_1 + M_2$ and $M_2 \preceq M_1 + M_2$. Therefore, $M_1 + M_2$ is a common refinement of M_1 and M_2. □

Theorem 5. Given 3-valued models M_1 and M_2, $M_1 \oplus M_2$ is their common abstraction.

Proof:
Proof follows from the proof of Theorem 6, where $\rho = \Sigma_1 \times \Sigma_2$. □

Theorem 6. Let ρ be a left/right total relation s.t. $(s_0,t_0) \in \rho$. Then, $M_1 \oplus M_2$ built using ρ is a common abstraction of M_1 and M_2.

Proof:
Similar to Theorem 4. Again, we proceed in two steps:

I. We first show that for every $s \in \Sigma_1$ if $(s,t) \in \rho$, then $(s,t) \preceq s$. It suffices to show $(s,t) \in \rho \Rightarrow (s,t) \preceq^i s$, for all $i \geq 0$. Proof is by induction on i:

Base case. $(s,t) \in \rho \Rightarrow (s,t) \preceq^0 s$.

$(s,t) \preceq^0 s$
$\Leftrightarrow$ (by the definition of $\preceq^0$)
$\forall p \in AP_u \cdot I_\oplus((s,t),p) \preceq I_1(s,p)$
$\Leftrightarrow$ (since $I_\oplus((s,t),p) = I_1(s,p) \sqcap I_2(t,p)$)
$\forall p \in AP_u \cdot I_1(s,p) \sqcap I_2(t,p) \preceq I_1((s,t),p))$
$\Leftrightarrow$ (by the properties of $\sqcup$)
true

Inductive case. Let $(s,t) \in \rho \Rightarrow (s,t) \preceq^n s$. We prove that

$$(s,t) \in \rho \Rightarrow (s,t) \preceq^{n+1} s$$

By Definition 10, we need to show:

1. $I_\oplus((s,t),p) \preceq I_1(s,p)$
2. $\forall (s',t') \in \Sigma_1 \times \Sigma_2 \cdot R_\oplus((s,t),(s',t')) \succeq \mathsf{t} \Rightarrow$
$\exists s' \in \Sigma_1 \cdot R_1(s,s') \succeq \mathsf{t} \wedge (s',t') \preceq^n s'$
3. $\forall s' \in \Sigma_1 \cdot R_1(s,s') \preceq \mathsf{t} \Rightarrow \exists (s',t') \in \Sigma_1 \times \Sigma_2 \cdot$
$R_\oplus((s,t),(s',t')) \preceq \mathsf{t} \wedge (s',t') \preceq^n t'$

1. From $I_{\oplus}((s,t),p) = I_1(s,p) \sqcap I_2(t,p)$, and $I_1(s,p) \sqcap I_2(t,p) \preceq I_1(s,p)$

2.

$$
\begin{array}{ll}
 & \forall (s',t') \in \Sigma_1 \times \Sigma_2 \cdot R_{\oplus}((s,t),(s',t')) \succeq \mathsf{t} \\
\Rightarrow & \text{(by the definition of } R_{\oplus}\text{)} \\
 & R_1(s,s') \sqcap R_2(t,t') \succeq \mathsf{t} \wedge (s',t') \in \rho \\
\Rightarrow & \text{(by the definition of } \sqcap\text{)} \\
 & \exists s' \in \Sigma_1 \cdot R_1(s,s') \succeq \mathsf{t} \wedge (s',t') \in \rho \\
\Rightarrow & \text{(by the inductive hypothesis)} \\
 & \exists s' \in \Sigma_1 \cdot R_1(s,s') \succeq \mathsf{t} \wedge (s',t') \preceq^n s'
\end{array}
$$

3.

$$
\begin{array}{ll}
 & \forall s' \in \Sigma_1 \cdot R_1(s,s') \preceq \mathsf{t} \\
\Rightarrow & \text{(since } \rho \text{ is left/right total)} \\
 & R_1(s,s') \preceq \mathsf{t} \wedge \exists t' \in \Sigma_2 \cdot (s',t') \in \rho \wedge R_2(t,t') \succeq \mathsf{m} \\
\Rightarrow & \text{(by the properties of } \sqcup\text{)} \\
 & \exists (s',t') \in \Sigma_1 \times \Sigma_2 \cdot R_1(s,s') \sqcap R_2(t,t') \preceq \mathsf{t} \wedge (s',t') \in \rho \\
\Rightarrow & \text{(by the definition of } R_{\oplus}\text{)} \\
 & \exists (s',t') \in \Sigma_1 \times \Sigma_2 \cdot R_{\oplus}((s,t),(s',t')) \preceq \mathsf{t} \wedge (s',t') \in \rho \\
\Rightarrow & \text{(by the inductive hypothesis)} \\
 & \exists (s',t') \in \Sigma_1 \times \Sigma_2 \cdot R_{\oplus}((s,t),(s',t')) \preceq \mathsf{t} \wedge (s',t') \preceq^n s'
\end{array}
$$

II. Similarly, we show that for every $t \in \Sigma_2$ if $(s,t) \in \rho$, then $(s,t) \preceq t$.

Since $(s_0,t_0) \in \rho$, by **I.** and **II.**, we obtained that $(s_0,t_0) \preceq s_0$ and $(s_0,t_0) \preceq t_0$. This implies that $M_1 \oplus M_2 \preceq M_1$ and $M_1 \oplus M_2 \preceq M_2$. Therefore, $M_1 \oplus M_2$ is a common abstraction of M_1 and M_2. $\square$