

On Consistency and Merge of Modal Transition Systems

Dario Fischbein
Department of Computing
Imperial College
180 Queen's Gate
London, SW7 2RH, UK
d.fischbein@doc.ic.ac.uk

Sebastian Uchitel
Department of Computer Science
CONICET - FCEN
University of Buenos Aires
Buenos Aires, C1428EGA, Argentina
suchitel@dc.uba.ar

ABSTRACT

In this paper we provide a complete characterization of MTS consistency and propose an algorithm for MTS merging that improves on state of the art.

1. INTRODUCTION

Labelled Transition Systems (LTSs) are convenient formalisms for modelling and reasoning about system behaviour at the architectural level. These models provide a basis for a wide range of automated analysis techniques, such as model-checking and simulation. LTS models are two valued in that they classify behaviour into two categories: those that are described by the LTS as paths or trees over its transitions and those that cannot be reproduced as paths or trees over its transitions. The former corresponds to the behaviour that the system is expected to provide while the latter is the behaviour that the system is expected not to provide.

The completeness assumption -knowing how to partition all behaviours into required and proscribed- is limiting in the context of software development process best-practices which include iterative development, adoption of use-case and scenario-based techniques and viewpoint- or stakeholder-based analysis; practices which require modelling and analysis in the presence of partial behaviour information.

Modal Transition Systems have been shown to be useful to reason about system behaviour in the context of partial information [5, 11, 3, 10]. MTS allow distinguishing the behaviour the system is expected to provide, the behaviour it is expected to never provide, and the behaviour for which it is still unknown or undecided whether it is acceptable or unacceptable behaviour. The semantics of a MTS can intuitively be thought of as a set of labelled transition systems. This set characterises all LTS that provide all the required behaviour described by the MTS, do not provide any of the proscribed behaviour described in the MTS, and have made some arbitrary decision on MTS unknown behaviour.

The notion of refinement [8] between MTS captures formally this intuition and provides an elegant way of describ-

ing the process of behaviour model elaboration as one in which behaviour information is acquired and introduced into the behaviour model incrementally, gradually refining an MTS until it characterises exactly one LTS.

A particularly useful notion in the context of software and requirements engineering is that of *merge*. Merging two consistent models is a process that should result in a minimal common refinement of both models where *consistency* is defined as the existence of one common refinement. Intuitively, merging builds a model that characterises the intersection of the LTS characterised by the models being merged. In other words, the merge characterises the LTS that provide all the required behaviour of the MTS being merged, and that do not provide any of the proscribed behaviour of the MTS being merged.

MTS merging can be used as the conjunction of multiple partial operational descriptions which may have been provided as MTS [11] or even synthesised from other description languages such as goal models and scenarios [10]. One of the current limitations of MTS merging is that a complete and correct algorithm for merging has not been developed. Larsen [7] originally proposed an algorithm that can only be applied to pairs of *independent* MTS models. Thus, there are MTS which have a least common refinement, for which the algorithm cannot be applied. Uchitel and Chechik [11] have proposed an algorithm, improved by Brunet [2], that can be applied to any consistent pair of MTS, but may not yield a minimal common refinement in some cases.

In this paper we provide a complete characterization of consistency (i.e. when a common refinement exists) and propose an algorithm for merging that improves both on [7] and [11]. The algorithm is guaranteed to always find a common refinement between two MTS if one exists. It is also guaranteed to produce a common refinement that is less refined than [7], [11], [2] if these were to return a common refinement. We believe that this algorithm can be improved to always deliver a minimal common refinement given two consistent models. In other words, that this is the basis for a correct and complete merge algorithm. We are currently experimenting with some promising improvements.

2. BACKGROUND

In this section, we review transition systems, fix notation and review refinement and consistency of MTSs.

LTSs describe the behaviour of a system by means of labelled transitions. The transitions determine for each state which actions are provided by the system and which are not. For the case in which an action is provided, the transition

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

system defines the system states that can result from taking the action. Figure 1.c shows an example of an LTS.

DEFINITION 1. (Labelled Transition Systems) *Let States be a universal set of states, Act be a universal set of observable action labels. A labelled transition system (LTS) is a tuple $P = (S, L, \Delta, s_0)$, where $S \subseteq \text{States}$ is a finite set of states, $L \subseteq \text{Act}$ a set of labels, $\Delta \subseteq (S \times L \times S)$ a transition relation between states, and $s_0 \in S$ the initial state. Given an LTS $P = (S, L, \Delta, s_0)$ we say P transitions on ℓ to P' , denoted $P \xrightarrow{\ell} P'$, if $P' = (S, L, \Delta, s'_0)$ and $(s_0, \ell, s'_0) \in \Delta$.*

MTSs extend LTSs by defining two sets of transitions. The first, similarly to LTS, describe the actions provided by the system in different states. The second set of transitions describes actions that may be provided by the system. If there is no transition from that a state on a particular action in either set of transitions, then the system will never provide the action on that state.

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. Given an MTS $M = (S, L, \Delta^r, \Delta^p, s_0)$ we say M transitions on ℓ through a required transition to M' , denoted $M \xrightarrow{\ell}_r M'$, if $M' = (S, L, \Delta^r, \Delta^p, s'_0)$ and $(s_0, \ell, s'_0) \in \Delta^r$, and M transitions through a possible transition, denoted $M \xrightarrow{\ell}_p M'$, if $(s_0, \ell, s'_0) \in \Delta^p$.*

We refer to transitions in $\Delta^p \setminus \Delta^r$ as *maybe* transitions. Maybe transitions are denoted with a question mark following the label. Note that LTS are a special type of MTS that do not have maybe transitions.

Strong refinement [8] 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 M if N preserves all of the required and all of the proscribed behaviours of M .

DEFINITION 3. (Strong Refinement Relation) *Let δ be the universe of all MTSs. A refinement relation R is a binary relation on δ such that if $(M, N) \in R$ then:*

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

DEFINITION 4. (Strong Refinement) *MTSN is a refinement of MTSN, denoted $M \preceq N$, if there exists a refinement relation R such that $(M, N) \in R$.*

The notion of implementation [6] relates an MTS with all the LTSs which have its required and forbidden behaviour. These LTSs may have taken some decisions over the maybe behaviour. This notion is important because allows interpreting an MTS in terms of its implementations. Note that a MTS with no maybe transitions can be interpreted as a LTS and that the notion of refinement when an MTS has no maybe transitions is LTS bisimulation [9].

DEFINITION 5. (Implementation) *We say that an LTS $I = (S_I, L_I, \Delta_I, i_0)$ is a implementation of an MTS $M = (S_M, L_M, \Delta_M^r, \Delta_M^p, m_0)$, written $M \preceq I$, if $M \preceq M_I$ with $M_I = (S_I, L_I, \Delta_I, \Delta_I, i_0)$. We also define the set of implementations of M as $\mathcal{I}[M] = \{I \mid M \preceq I\}$.*

Intuitively two models that provide partial descriptions of the same system are consistent [11] if the required and forbidden behaviour described by each of them are compatible. In other words, consistency is defined as the existence of a common refinement.

DEFINITION 6. (Consistency) *Two MTSs M and N are consistent if there exists an LTS I such that I is a common implementation of M and N .*

Note that this definition is equivalent to saying that two MTSs M and N are consistent if there exists an MTS C such that C is a common refinement of M and N .

3. MERGING MTS

In this section, we present the main results of this paper. We first recall the definition of MTS merging [7, 11], we then present a complete characterization of consistency, the pre-condition for merge, and finally present a merge algorithm that improves on existing ones [7, 11, 2].

The intuition captured by merge 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. The notion of refinement underlies this intuition as it captures the “more defined than” relation between two partial models: merging should result in the least refined common refinement of the models being merged.

DEFINITION 7. (Common Refinement) *A modal transition system P is a common refinement of modal transition systems M and N if $M \preceq P$ and $N \preceq P$.*

DEFINITION 8. (Least Common Refinement) *A modal transition system P is the least common refinement (LCR) of modal transition systems M and N if P is a common refinement of M and N , and for any common refinement Q of M and N , $P \preceq Q$.*

3.1 Checking Consistency

Checking if two models are consistent is of clear use to engineers that have multiple partial descriptions of system behaviour. In particular, consistency is a pre-condition for merging models as there can be no most abstract common refinement if there are no common refinements. In this section we first analyse the *independence* notion defined in [7], which intuitively was defined to capture when two models are not contradictory and is used to define merge (called conjunction in [7]) on MTSs. We show that independence does not characterise when two models can be merged. We then present a complete characterisation of consistency and an algorithm for determining consistency.

DEFINITION 9. (Independence [7]) *An independence relation R is a binary relation on δ such that if $(S, T) \in R$ then:*

1. $(\forall \ell, S')(S \xrightarrow{\ell}_r S' \implies (\exists! T')(T \xrightarrow{\ell}_p T' \wedge (S', T') \in R))$
2. $(\forall \ell, T')(T \xrightarrow{\ell}_r T' \implies (\exists! S')(S \xrightarrow{\ell}_p S' \wedge (S', T') \in R))$
3. $(\forall \ell, S', T')(S \xrightarrow{\ell}_p S' \wedge T \xrightarrow{\ell}_p T') \implies (S', T') \in R$

Consider models \mathcal{A} and \mathcal{B} of Fig. 1 which are not independent: if $(0, 0)$ were in the independence relation then $(1, 1)$ must be as well because of rule 3. However, rule 1 would be violated because model \mathcal{B} can do a required action on b from state 1 but model \mathcal{A} cannot follow this action with a possible b . Therefore $(1, 1)$ cannot be in the relation, which implies that $(0, 0)$ cannot be in it either. Although these models are not independent they are consistent since they have a common refinement \mathcal{C} .

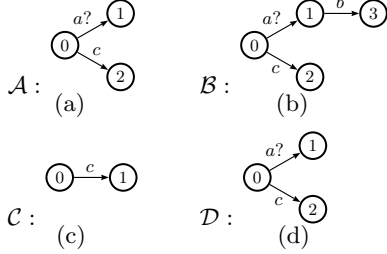


Figure 1:

We now define a new relation, *strong consistency relation*, and show that it characterises consistency, i.e. we prove that two models are consistent if and only if they can be related by the strong consistency relation.

DEFINITION 10. (Strong Consistency Relation) A strong consistency relation is a binary relation $C \subseteq \delta \times \delta$, such that the following conditions hold for all $(M, N) \in C$:

1. $(\forall \ell, M')(M \xrightarrow{\ell}_r M' \implies (\exists N')(N \xrightarrow{\ell}_p N' \wedge (M', N') \in C))$
2. $(\forall \ell, N')(N \xrightarrow{\ell}_r N' \implies (\exists M')(M \xrightarrow{\ell}_p M' \wedge (M', N') \in C))$

Intuitively, this relation requires that one model provides as possible behaviour at least all the required behaviour of the other, and vice versa. For example, $R = \{(0, 0), (2, 2)\}$ is a strong consistency relation between models \mathcal{A} and \mathcal{B} . We can see that the strong consistency relation is weaker than the independence relation since it does not have condition 3 of the independence relation and conditions 1 and 2, although similar to those of the independence relation, do not require a deterministic choice. This weaker relation relates models which were related by independence but also relates pairs of models that are not independent, such as \mathcal{A} and \mathcal{B} . Furthermore the strong consistency relation only relates those models which are consistent.

THEOREM 1. (Strong Consistency Relation Characterizes Consistency) Two MTSs M and N are consistent if and only if there exists a strong consistency relation C_{MN} such that (M, N) is contained in C_{MN} .

PROOF. \Leftarrow Let CI be a LTS defined by $CI = (C_{MN}, Act, \Delta_{CI}, (M_0, N_0))$ where Δ_{CI} is the smallest relation that satisfies the following rules, assuming that $\{(M, N), (M', N') \subseteq C_{MN}\}$.

$$RP \frac{M \xrightarrow{\ell}_r M', N \xrightarrow{\ell}_p N'}{(M, N) \xrightarrow{\ell}_r (M', N')} \quad PR \frac{M \xrightarrow{\ell}_p M', N \xrightarrow{\ell}_r N'}{(M, N) \xrightarrow{\ell}_p (M', N')}$$

It is easy to prove that $M \preceq CI$ using that $R = \{(M, (M, N)) \mid (M, N) \in C_{MN}\}$ is an implementation relation between M and CI .

\Rightarrow) Since M and N are consistent we can take an LTS CI such that $M \preceq CI$ and $N \preceq CI$. By definition of strong semantics there exist R_M and R_N implementation relations between M and CI , and between N and CI respectively.

Let C_{MN} be a relation defined by $C_{MN} = R_M \circ R_N^{-1}$. It can easily be proven that C_{MN} is a strong consistency relation between M and N . \square

We have developed a fixed point algorithm for checking consistency that starts with the Cartesian product of the states and iteratively eliminates the pairs that are not valid according to the strong consistency relation. The completeness and correctness proofs for this algorithm are straightforward, a proof of a similar algorithm can be found in [4].

3.2 Computing Merge

Although merging is precisely defined, a complete and correct algorithm to compute this operation has not been developed. In this section, we first analyse two different operations defined in [7, 2] as approximations for computing the merge. Finally, we present a novel algorithm which, although it does not always return the optimal solution, it always returns a common refinement for consistent models and the result is an equivalent or better solution than [2] and [7] if this were to return a common refinement.

3.2.1 Limitations of Existing Algorithms

In [7] an operation between two models called conjunction is defined. This operation when applied to independent models gives their LCR.

DEFINITION 11. (Conjunction) [7] Let M and N be MTSs, the conjunction of M and N is defined as $M \wedge N = (S_M \times S_N, L, \Delta_{M \wedge N}^r, \Delta_{M \wedge N}^p, (m_0, n_0))$, where $\Delta_{M \wedge N}^r, \Delta_{M \wedge N}^p$ are the smallest relations which satisfy the following rules:

$$RP \frac{M \xrightarrow{\ell}_r M', N \xrightarrow{\ell}_p N'}{(M, N) \xrightarrow{\ell}_r (M', N')} \quad PR \frac{M \xrightarrow{\ell}_p M', N \xrightarrow{\ell}_r N'}{(M, N) \xrightarrow{\ell}_p (M', N')}$$

$$PP \frac{M \xrightarrow{\ell}_p M', N \xrightarrow{\ell}_p N'}{(M, N) \xrightarrow{\ell}_p (M', N')}$$

The limitation of the conjunction operator is that there are models with an LCR that the operator fails to produce. In fact, in these cases, the operator does not produce a common refinement. Consider models \mathcal{A} and \mathcal{B} of Fig. 1. The LCR of these models is \mathcal{C} , however their conjunction is model \mathcal{D} (all depicted in Fig. 1). This problem occurs when two models are not independent but they are consistent.

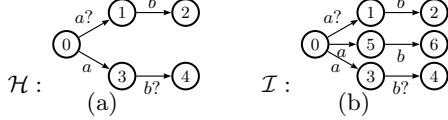
To overcome the problems of the conjunction operator a variation of the conjunction operator was proposed (initially in [11] and then improved in [2]). These operators generalise the problem addressed in [7] to support merging models with different alphabets under weak (i.e. observational) refinement. When the models being merged have the same alphabet and no unobservable actions, the problem is that of merge under strong semantics as discussed in this paper. Hence, in this paper we analyse the operator in [2] restricted to models with the same alphabet and no unobservable actions under strong semantics.

DEFINITION 12. (The $+_{cr}$ operator) Let M and N be MTSs and let C_{MN} be the largest strong consistency relation between them. The $+_{cr}$ operator between M and N is defined as $M +_{cr} N = (C_{MN}, L, \Delta_{M +_{cr} N}^r, \Delta_{M +_{cr} N}^p, (m_0, n_0))$, where

$\Delta_{M+_{cr}N}^r, \Delta_{M+_{cr}N}^p$ are the smallest relations which satisfy rules *RP*, *PR*, *PP* of Def. 11:

Note that the difference between the conjunction and the $+_{cr}$ operator is that the latter only considers pairs of states which are consistent. Returning to the example on Fig. 1, the consistency relation between models \mathcal{A} and \mathcal{B} is $\{(0,0), (2,2)\}$, therefore if we apply the $+_{cr}$ to these models the rules that can be applied are *PR* or *RP* producing a required transition between the state $(0,0)$ and $(2,2)$ by c . Thus the result of $\mathcal{A}+_{cr}\mathcal{B}$ is the model \mathcal{C} , which is the expected merge.

THEOREM 2. *If M and N are consistent MTSs then $M+_{cr}N$ is always a common refinement of M and N .*



We now show an example of when the above operator does not produce an LCR. Clearly the merge of a model with itself should result in the same model (i.e. merge is idempotent) as any model is a refinement of itself and is the least refined as possible. Consider the following example depicted in Fig. 3.2.1: The result of $\mathcal{H}+_{cr}\mathcal{H}$ is \mathcal{I} , which is more refined than \mathcal{H} . Note that the conjunction operator produces the same result.

The problem exemplified above arises because $+_{cr}$ does not deal correctly with nondeterminism when there is a mix of required and maybe transitions. Under this circumstance the $+_{cr}$ will apply rules *RP* and *PR*, which guarantee to produce a *CR* but might fail to produce the *LCR*.

The point is that the rules *RM* and *MR* of [11, 2] take a conservative decision on how to merge a required with a maybe transition, i.e. that in some cases these rules create a required transition when a maybe transition would suffice. The algorithm we present below is based on detecting the required transitions resulting from these conservative rules which can be converted to maybe transitions.

3.2.2 A New Merge Algorithm

The algorithm we propose iteratively abstracts the result of $M+_{cr}N$ by replacing required transitions with maybe transitions. It does so while guaranteeing that the resulting MTS after each iteration continues to be a refinement of M and N . The decision of which transitions can be replaced is done by analysing all outgoing required transitions from a given state on a given label. The key notion here is that of *Cover Set*. Intuitively a cover set describes a set of outgoing required transitions from a given state and on a given label such that if we only keep these as required the model continues to be a common refinement of M and N .

Technically the definition of *cover set* defines, given a state c and a label ℓ , a set of states reachable from c on required transitions labelled ℓ such that the required transitions can simulate behaviour starting with ℓ from c .

DEFINITION 13. (*Cover Set*) *Let A, B, C be MTSs, R_{AC}, R_{BC} be refinement relations between A and C , and B and C respectively. Given $C_i \in S_C$ and $\ell \in Act$ we define a cover*

set over C_i on ℓ as a set $\zeta_{C_i,\ell}$ of states of C for which the following holds:

1. $\zeta_{C_i,\ell} \subseteq \Delta_C^r(C_i, \ell)$
2. $\Delta_A^r(R_{AC}^{-1}(C_i), \ell) \subseteq R_{AC}^{-1}(\zeta_{C_i,\ell})$
3. $\Delta_B^r(R_{BC}^{-1}(C_i), \ell) \subseteq R_{BC}^{-1}(\zeta_{C_i,\ell})$

Notation: $\Delta_r(S, \ell) = \{t \mid s \xrightarrow{\ell} t \wedge s \in S\}$

As an example of cover set, consider \mathcal{I} . The cover sets for state 0 and label a are the following: $\{5\}$, $\{3\}$ and $\{3, 5\}$ (these sets come from consider the following set of transitions $\{0 \xrightarrow{a} 5\}$, $\{0 \xrightarrow{a} 3\}$, and $\{0 \xrightarrow{a} 3, 0 \xrightarrow{a} 5\}$).

The merge algorithm we propose identifies a cover set for each state s and label ℓ of $A+_{cr}B$ and replaces any required transitions from s on ℓ that is not in the cover set with a maybe transition. We call this MTS transformation and abstraction operation and formalise it as follows:

DEFINITION 14. (*Abstraction operation*) *Let A, B, C be MTSs as in Def. 13. Given a cover set, $\zeta_{C_i,\ell}$, over C_i on ℓ we define the following operation:*

$\mathcal{A}(C, \zeta_{C_i,\ell}) = (S_C, \Delta_C^p, \Delta_C^r, c_0)$ where Δ_C^r is defined by $\Delta_C^r = \Delta_C^r - \{(c_i, \ell, c') \mid c' \notin \zeta_{C_i,\ell}\}$

It is straightforward to show that the abstraction operation on models A, B , and C effectively produces an abstraction of C . However, it is also the case that it produces a common refinement of A and B .

THEOREM 3. (*Abstraction operation produces a CR*) $A \preceq \mathcal{A}(C, \zeta_{C_i,\ell})$ and $B \preceq \mathcal{A}(C, \zeta_{C_i,\ell})$ for all cover set $\zeta_{C_i,\ell}$

PROOF. *We want to prove that R_{AC} is a refinement relation between A and $\mathcal{A}(C, \zeta_{C_i,\ell})$. Suppose that this is false, therefore there exists a pair (A', C') in R_{AC} that does not fulfil the refinement relation conditions. If so, we have one possible transition of C' in $\mathcal{A}(C, \zeta_{C_i,\ell})$ that A' cannot simulate; or A' has one required transition, which C' in $\mathcal{A}(C, \zeta_{C_i,\ell})$ cannot simulate with a required transition. The first of these two cases is impossible because $\mathcal{A}(C, \zeta_{C_i,\ell})$ has the same possible transitions as C , and A simulates all possible transitions of C . Therefore the latter must be the case.*

Consequently there exists a required transition, $A' \xrightarrow{\ell'} A''$, that cannot be simulated by C' in $\mathcal{A}(C, \zeta_{C_i,\ell})$, i.e. $\exists C'' \cdot C' \xrightarrow{\ell'} C'' \wedge (A'', C'') \in R_{AC}$. Since R_{AC} is a refinement relation between A and C , and C and $\mathcal{A}(C, \zeta_{C_i,\ell})$ only differ on the required transitions of C_i on ℓ , then C' must be C_i and ℓ' must be ℓ . Then $(A', C_i) \in R_{AC}$ consequently $A' \in R_{AC}^{-1}(C_i)$, and by the fact that $A' \xrightarrow{\ell'} A''$, it follows that $A'' \in \xrightarrow{\ell}_r (R_{AC}^{-1}(C_i))$. Then by definition of cover set it follows that $A'' \in R_{AC}^{-1}(\zeta_{C_i,\ell})$. This implies that $\exists C'' \cdot (A'', C'') \in R_{AC} \wedge C' \xrightarrow{\ell'} C''$ in $\mathcal{A}(C, \zeta_{C_i,\ell})$ is a contradiction, which comes from the first assumption that R_{AC} is not a refinement relation between A and $\mathcal{A}(C, \zeta_{C_i,\ell})$. As a result we have proved that R_{AC} is a refinement relation between A and $\mathcal{A}(C, \zeta_{C_i,\ell})$.

Analogously it can be proven that $B \preceq \mathcal{A}(C, \zeta_{C_i,\ell})$. \square

Given that many different cover sets may exist for a specific state and label, the algorithm exploits the following

results to select which cover set to use as the basis for applying an abstraction operation and consequently producing a more abstract common refinement. We first define a refinement relation between cover sets which describes a partial order of cover sets based how refined the application of the abstraction operation using each cover set is. In other words, a cover set is more refined than another if the abstraction operation using the former yields a more refined model than applying the abstraction operation on the latter. We then prove that a refinement between cover sets can be established by checking if all states in one of the cover sets is refined by some state in the other cover set. This property provides a way computing which cover set to use when applying the abstraction operation described above.

DEFINITION 15 (COVER SET REFINEMENT). *Given $\zeta_{C_i,\ell}$ and $\zeta'_{C_i,\ell}$ cover sets over C_i on ℓ we say that $\zeta_{C_i,\ell}$ is refined by $\zeta'_{C_i,\ell}$, written $\zeta_{C_i,\ell} \preceq \zeta'_{C_i,\ell}$, iff $\mathcal{A}(C, \zeta_{C_i,\ell}) \preceq \mathcal{A}(C, \zeta'_{C_i,\ell})$. Also we might say that $\zeta_{C_i,\ell}$ is more abstract than $\zeta'_{C_i,\ell}$.*

PROPERTY 1. *Given $\zeta_{C_i,\ell}$ and $\zeta'_{C_i,\ell}$ cover sets over C_i on ℓ , $\zeta_{C_i,\ell}$ is refined by $\zeta'_{C_i,\ell}$ if the following condition holds: $\forall C_1 \in \zeta_{C_i,\ell} \cdot \exists C_2 \in \zeta'_{C_i,\ell} \cdot C_1 \preceq C_2$*

We now present the merge algorithm, which starting from $A +_{cr} B$ iteratively applies abstraction operations by identifying for each state and label its least refined cover set.

ALGORITHM 1 (MERGE ALGORITHM).

1. $M \leftarrow A +_{cr} B$, $isLCR \leftarrow true$
2. For each $(x, y) \in S_M$ and each $\ell \in Act$ do
 - 2.1 Get most abstract minimal cover set of (x, y) on ℓ .
 - 2.2 If minimal not unique, choose any and $isLCR \leftarrow false$.
 - 2.3 $M \leftarrow \mathcal{A}(M, \zeta_{(x,y),\ell})$
3. Return $(M, isLCR)$

From the results described above it follows that the merge algorithm produces a common refinement of A and B .

THEOREM 4. *The Algorithm 1 produces a common refinement of A and B .*

PROOF. *Follows from the fact that $A +_{cr} B$ is a common refinement of A and B and Theorem 3. \square*

3.2.3 Comparison with Existing Merge Algorithms

It is simple to show that the proposed algorithm produces a less refined merge than [11, 2] as the algorithm starts from the result of $+_{cr}$ and applies zero or more abstraction operations. This entails that the results that guarantee computing LCR for $+_{cr}$ also apply for the merge algorithm presented herewith. For the case of deterministic MTS, our algorithm produces the same result as [11, 2], in fact, this is also the case for models that satisfy the non-determinacy condition (a slightly weaker condition presented in [11]). This is due to the fact that the non-determinacy condition guarantees that the cover set for every state s and label ℓ contains all required transitions from s on ℓ , and consequently, the abstraction operation becomes idempotent.

The algorithm proposed generalizes that of [7] in the sense that it results in the same LCR for any pair of independent MTS, but can also be applied to non-independent yet consistent MTS producing a common refinement that in many cases is the LCR.

4. CONCLUSIONS AND FUTURE WORK

In this paper we have provided a complete characterization of MTS consistency and proposed an algorithm for merging that improves both on [7] and [11]. An implementation of merge and consistency check are provided in the MTS analyser (<http://lafhis.dc.uba.ar/~suchitel/MTSA.html>). Future work involves a characterization of the conditions under which the merge algorithm effectively produces an LCR. In addition, we believe that further improvements to the algorithm may yield a complete merge algorithm. We have preliminary experimental results of the latter and are currently working on the proofs. We also aim to study an adaptation of this algorithm for weaker semantic notions such as branching [5] and weak refinement [11].

5. REFERENCES

- [1] E. Brinksma, R. Cleaveland, K. G. Larsen, T. Margaria, and B. Steffen, editors. *Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Aarhus, Denmark, May 19-20, 1995, Proceedings*, volume 1019 of *Lecture Notes in Computer Science*. Springer, 1995.
- [2] G. Brunet. "A Characterization of Merging Partial Behavioural Models". Master's thesis, Univ. of Toronto, January 2006.
- [3] G. Brunet, M. Chechik, and S. Uchitel. Properties of behavioural model merging. In J. Misra, T. Nipkow, and E. Sekerinski, editors, *FM*, volume 4085 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2006.
- [4] D. Fischbein. Branching semantics for modal transition systems. Master's thesis, University of Buenos Aires, Department of Computing, April 2006.
- [5] D. Fischbein, S. Uchitel, and V. Braberman. A foundation for behavioural conformance in software product line architectures. In *ROSATEA '06: Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis*, pages 39–48, New York, NY, USA, 2006. ACM Press.
- [6] M. Huth. Refinement is complete for implementations. *Formal Asp. Comput.*, 17(2):113–137, 2005.
- [7] K. G. Larsen, B. Steffen, and C. Weise. A constraint oriented proof methodology based on modal transition systems. In Brinksma et al. [1], pages 17–40.
- [8] K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210. IEEE Computer Society, 1988.
- [9] R. Milner. A modal characterisation of observable machine-behaviour. In *CAAP '81: Proceedings of the 6th Colloquium on Trees in Algebra and Programming*, pages 25–34, London, UK, 1981. Springer-Verlag.
- [10] S. Uchitel, G. Brunet, and M. Chechik. Behaviour model synthesis from properties and scenarios. In *ICSE*, pages 34–43. IEEE Computer Society, 2007.
- [11] S. Uchitel and M. Chechik. Merging partial behavioural models. In R. N. Taylor and M. B. Dwyer, editors, *SIGSOFT FSE*, pages 43–52. ACM, 2004.