A Sound Observational Semantics for Modal Transition Systems

Dario Fischbein[1], Victor Braberman[2], and Sebastian Uchitel[1,2]

[1] Imperial College London, 180 Queen's Gate, London, SW7 2RH, UK
[2] University of Buenos Aires, C1428EGA, Argentina
d.fischbein@doc.ic.ac.uk, {suchitel,vbraber}@dc.uba.ar

Abstract. Modal Transition Systems (MTS) are an extension of Labelled Transition Systems (LTS) that distinguish between required, proscribed and unknown behaviour and come equipped with a notion of refinement that supports incremental modelling where unknown behaviour is iteratively elaborated into required or proscribed behaviour. The original formulation of MTS introduces two alternative semantics for MTS, strong and weak, which require MTS models to have the same communicating alphabet, the latter allowing the use of a distinguished unobservable action. In this paper we show that the requirement of fixing the alphabet for MTS semantics and the treatment of observable actions are limiting if MTS are to support incremental elaboration of partial behaviour models. We present a novel semantics, branching alphabet semantics, for MTS inspired by branching LTS equivalence, we show that some unintuitive refinements allowed by weak semantics are avoided, and prove a number of theorems that relate branching refinement with alphabet refinement and consistency. These theorems, which do not hold for other semantics, support the argument for considering branching implementation of MTS as the basis for a sound semantics to support behaviour model elaboration.

1 Introduction

Labelled Transition Systems [13] (LTS) have been used successfully to reason about system behaviour. Modal Transition Systems [16] (MTS) are an extension of LTS that distinguish between required, proscribed and unknown behaviour. MTS have been studied for some time as a means for formally describing partial knowledge of the intended behaviour of software systems.

An MTS can be naturally interpreted as the set of implementations, in the form of LTS, that conform to the MTS. Hence, with a view to support elaboration of partial behaviour models operations over MTS and the implementations they describe have been studied. These include refinement [1,11,19] (does an MTS describe a subset of the implementations of another MTS?), consistency [19,7] (is the intersection of implementations described by two MTS non-empty?) and merge [15,7,19] (which are the implementations that conform to two MTS?).

The original formulation of MTS by Larsen [16] defined two semantics by presenting two refinement relations between MTS. The first, strong refinement, requires MTS to have the same alphabet, i.e. the same set of transition labels, the second, weak refinement, allows the use of a distinguished unobservable action as in, for instance, process algebraic approaches to behaviour modelling.

Although strong semantics for MTS has a number of convenient qualities [7,16], the requirement of a fixed set of action labels and the inability to distinguish observable from non-observable actions results in a serious limitation for using MTS as the basis for behaviour model elaboration: Incremental elaboration typically involves gradually extending the scope of a description (i.e. augmenting the alphabet of MTS) and also merging models with different scopes.

Weak semantics for MTS supports the distinction between observable and non-observable actions, hence when combined with hiding operations, MTS under weak semantics supports a variety of elaboration tasks including merge [19,3]. However, as we show in this paper, this semantics allows some counter-intuitive LTS implementations and lacks some expected theoretical properties. In particular, it does not behave as expected with respect to alphabet hiding.

In this paper we discuss the limitations of existing semantics for MTS and propose a novel semantics, inspired by the notion of branching equivalence and branching simulation [21,9] for LTS, that addresses these limitations. More specifically, we present branching semantics for MTS and define notions of branching implementation and branching alphabet implementation. We show that unintuitive implementations allowed by weak semantics are avoided by branching semantics and prove a number of theorems that relate branching refinement with alphabet extension that do not hold for weak semantics. In addition,we study the notion of consistency, a key notion in the context of partial behaviour model elaboration, and show results for branching semantics that do not hold for weak semantics, thus, further supporting the argument for considering branching implementation of MTS as the basis for a sound semantics to support behaviour model elaboration.

2 Background

In this section, we recall definitions and fix notation for Labelled Transition Systems, related equivalences, and Modal Transition Systems.

Labelled transition systems (LTSs) [13] are widely used for modelling and analysing the behaviour of software systems. An LTS is a state transition system where transitions are labelled with actions. The set of actions of an LTS is called its *communicating alphabet* and constitutes the interactions that the modelled system can have with its environment. In addition, LTSs can have transitions labelled with τ, representing actions that are not observable by the environment. Figure 2 shows an example of an LTS.

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

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 $(s_0, \ell, s'_0) \in \Delta$. Similarly, we write $P \xrightarrow{\hat{\ell}} P'$ to denote that either $P \xrightarrow{\ell} P'$ or $\ell = \tau$ and $P = P'$ are true. We use $P \overset{\ell}{\Longrightarrow} P'$ to denote $P(\xrightarrow{\tau})^* \xrightarrow{\ell} (\xrightarrow{\tau})^* P'$, and $P \overset{\hat{\ell}}{\Longrightarrow} P'$ to denote $P(\xrightarrow{\tau})^* \xrightarrow{\hat{\ell}} (\xrightarrow{\tau})^* P'$.

A number of equivalence relations have been proposed that provide a criteria for deciding if syntactically different LTS models describe the same behaviour.

Definition 2. (Strong Bisimulation Equivalence) *Let $\wp$ be the universe of all LTS, and $P, Q \in \wp$. P and Q are strong equivalent, written $P \sim Q$, if $\alpha P = \alpha Q$ and (P, Q) is contained in some bisimulation relation $R \subseteq \wp \times \wp$ for which the following holds for all $\ell \in Act_\tau$:*

$$1.\ (P \xrightarrow{\ell} P') \implies (\exists Q' \cdot Q \xrightarrow{\ell} Q' \ \wedge\ (P', Q') \in R)$$
$$2.\ (Q \xrightarrow{\ell} Q') \implies (\exists P' \cdot P \xrightarrow{\ell} P' \ \wedge\ (P', Q') \in R)$$

This equivalence does not distinguish τ as special or unobservable actions. A property of this equivalence is that it preserves the branching structure of processes [9]. In contrast Weak Bisimulation equivalence compares the observable behaviour of models and ignores internal computations (τ-transitions). Some authors call this equivalence *observational equivalence*, but we use this expression to refer to any equivalence that considers τ-transitions as unobservable actions.

Definition 3. (Weak Bisimulation Equivalence) *Let $\wp$ be the universe of all LTS, and $P, Q \in \wp$. P and Q are weak bisimulation equivalent, written $P \approx_{\mathrm{w}} Q$, if $\alpha P = \alpha Q$ and (P, Q) is contained in some weak bisimulation relation $R \subseteq \wp \times \wp$ for which the following holds for all $\ell \in Act_\tau$:*

$$1.\ (P \xrightarrow{\ell} P') \implies (\exists Q' \cdot Q \overset{\hat{\ell}}{\Longrightarrow} Q' \ \wedge\ (P', Q') \in R)$$
$$2.\ (Q \xrightarrow{\ell} Q') \implies (\exists P' \cdot P \overset{\hat{\ell}}{\Longrightarrow} P' \ \wedge\ (P', Q') \in R)$$

Finally, branching equivalence is the coarsest observational equivalence that preserves the branching structure of processes [9], it is coarser than strong equivalence yet finer than weak bisimulation equivalence.

Definition 4. (Branching Bisimulation Equivalence)

Let $\wp$ be the universe of all LTS, and $P, Q \in \wp$. P and Q are branching bisimulation equivalent, written $P \approx_{\mathrm{b}} Q$, if $\alpha P = \alpha Q$ and (P, Q) is contained in some observational bisimulation relation $R \subseteq \wp \times \wp$ for which the following holds for all $\ell \in Act_\tau$:

$$1.\ (P \xrightarrow{\ell} P') \implies (\exists Q', Q'' \cdot Q \overset{\hat{\tau}}{\Longrightarrow} Q' \xrightarrow{\hat{\ell}} Q'' \ \wedge\ (P, Q') \in R \ \wedge\ (P', Q'') \in R)$$
$$2.\ (Q \xrightarrow{\ell} Q') \implies (\exists P', P'' \cdot P \overset{\hat{\tau}}{\Longrightarrow} P' \xrightarrow{\hat{\ell}} P'' \ \wedge\ (P', Q) \in R \ \wedge\ (P'', Q') \in R)$$

MTSs [16] 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 5. (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 (resp. possible) transition to M', denoted $M \xrightarrow{\ell}_{\mathrm{r}} M'$ (resp. $M \xrightarrow{\ell}_{\mathrm{p}} M'$), if $M' = (S, L, \Delta^r, \Delta^p, s_0')$ and $(s_0, \ell, s_0') \in \Delta^r$ (resp. $(s_0, \ell, s_0') \in \Delta^p$).

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 case of MTS where there are no maybe transitions.

3 Motivation

In this section we analyse the adequacy of existing MTS semantics for incremental modelling of system behaviour using a simple motivating example.

3.1 Motivating Example

Consider a behaviour model of the control software for an electronic device at an early stage of the modelling process. The device offers different functions grouped into several menus. The general behaviour of the system is basically as follows: the user selects a desired menu and the system offers the functions associated with the menu. If the user does not choose any function after an elapsed time, the system beeps and returns to the initial state. The MTS that models the controller's behaviour is shown in Figure 1. Note that the model abstracts away using τ transitions how the functionality selected by a user works. From the initial state there are n transitions labelled $menu_1$ to $menu_n$ each one representing the selection of a menu by the user. These transitions are either required or maybe, the former corresponding to the menu items that must be in the final product and the latter corresponding to those whose inclusion is still in doubt. States labelled M_i model that the user has selected the menu i and that a functions $func_1$ to $func_{xi}$ are available. The user can select one of these functions and the system will do the associated task and and then return to the initial state, or an internal timeout occurs, making the system leave the M_i state and return to the initial state with a *beep*. This timeout is an internal event and therefore not visible to the user, so it has been modelled with a τ transition.

The explanation given above for Figure 1, although intuitive, is informal. We now discuss its precise meaning by recalling existing semantics for MTS.

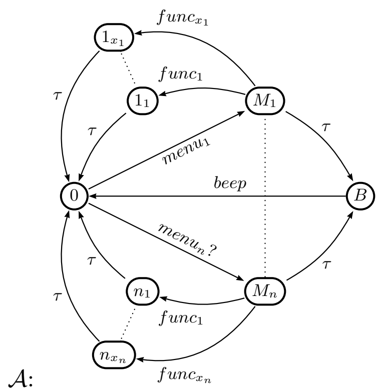

Fig. 1. MTS for Controller

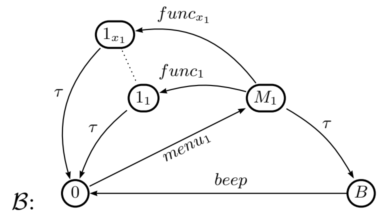

Fig. 2. A strong refinement of Figure 1 where only $menu_1$ is available

3.2 Strong Semantics

Strong refinement [16] of MTS 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 MTS 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. Alternatively, an MTS N refines M if N can simulate the required behaviour of M, and M can simulate the possible behaviour of N.

Definition 6 (Strong Refinement). *[16] Let δ be the universe of all MTS. N is a refinement of M, written $M \preceq N$, if $\alpha M = \alpha N$ and (M, N) is contained in some refinement relation $R \subseteq \delta \times \delta$ for which the following holds for all $\ell \in Act_\tau$:*

1. $(M \xrightarrow{\ell}_{\mathrm{r}} M') \implies (\exists N' \cdot N \xrightarrow{\ell}_{\mathrm{r}} N' \;\wedge\; (M', N') \in R)$
2. $(N \xrightarrow{\ell}_{\mathrm{p}} N') \implies (\exists M' \cdot M \xrightarrow{\ell}_{\mathrm{p}} M' \;\wedge\; (M', N') \in R)$

Note that *strong refinement* for MTS does not distinguish τ as an unobservable action and is equivalent to strong bisimulation when restricted to LTS models.

Consider the MTS shown in Figure 1. If modellers decide to exclude $menu_n$ then the model that would represent that decision is the one shown in Figure 2. According to strong semantics this latter model is a valid possible evolution of the initial one since the MTS $\mathcal{A}$ is refined by the MTS $\mathcal{B}$ ($\mathcal{A} \preceq \mathcal{B}$), incorporating as new knowledge that the $menu_n$ has been removed from the functionalities of the system. The refinement relation between these models is $R = \{(0, 0), (B, B), (M_1, M_1), (1_1, 1_1), \ldots, (1_{x_1}, 1_{x_1})\}$.

Note the MTS $\mathcal{B}$ in Figure 2 has no maybe transitions, thus it can be considered an LTS. We say that it is an implementation of the model in Figure 1.

Definition 7 ((Strong) Implementation). *We say that an LTS $I = (S_I, L_I, \Delta_I, i_0)$ is a* (strong) implementation *of an MTS $M = (S_M, L_M, \Delta^r_M, \Delta^p_M, m_0)$, 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 $\mathcal{I}[M] = \{I \text{ LTS} \mid M \preceq I\}$.*

In fact, we shall consider the *strong semantics* of an MTS as its set of strong implementations and interpret strong refinement as the partial order determined by the subset relation over sets of strong implementations. Note that Larsen's strong refinement relation is transitive [16] and therefore it is straightforward to proof that $M \preceq M'$ implies $\mathcal{I}[M] \supseteq \mathcal{I}[M']$, which means that the $\preceq$ relation is of great use to reason efficiently about elaborating partial models. Although it was thought that $\mathcal{I}[M] \supseteq \mathcal{I}[M'] \Leftrightarrow M \preceq M'$ [10] this is not the case [6].

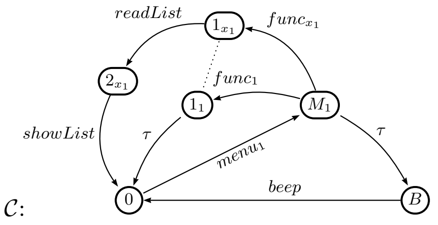

Fig. 3. A model where the behaviour of functionality associated to $func_{x_1}$ has been detailed

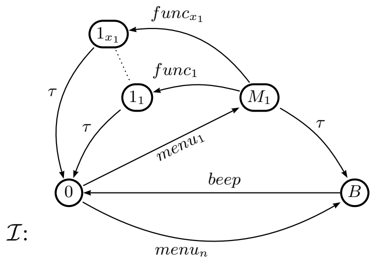

Fig. 4. A valid implementation of the initial model according to weak refinement

Strong semantics does not adequately support iterative model elaboration because in practice such an activity often requires progressively extending the alphabet of the system to describe behaviour aspects that previously had not been taken into account. For instance, we may want to produce a model for the electronic device's controller which describes in more detail how a particular function works (see Figure 3, states 1_{x_1} and 2_{x_1}), and then check if this model conforms to the initial, more abstract model of the controller. Such check cannot be done with strong semantics as the models have different alphabets. A standard workaround for checking if Figure 3 conforms to Figure 1 is to hide actions *readList* and *showList* (i.e. replace them with τ) to obtain models with the same alphabet and then comparing them. Strong refinement is not appropriate in this case as it does not consider τ transitions as unobservable. Indeed, the model obtained by hiding *readList* and *showList* in Figure 3 is not a strong refinement of Figure 1. However, these models can be compared using an observational semantics. We discuss this below.

3.3 Weak Semantics

Weak MTS refinement also defined by Larsen [11] allows comparing the observable behaviour of models while ignoring the possible differences that they may have in terms of internal computation. In other words, this notion of refinement considers τ-labelled transitions differently from other transitions.

Definition 8 (Weak Refinement). *[11] N is a weak refinement of M, written $M \preceq_{\mathrm{w}} N$, if $\alpha M = \alpha N$ and (M, N) is contained in some refinement relation $R \subseteq \delta \times \delta$ for which the following holds for all $\ell \in Act_\tau$:*

$$1.\ (M \xrightarrow{\ell}_{\mathrm{r}} M') \implies (\exists N' \cdot N \overset{\hat{\ell}}{\Longrightarrow}_{\mathrm{r}} N' \ \wedge\ (M', N') \in R)$$
$$2.\ (N \xrightarrow{\ell}_{\mathrm{p}} N') \implies (\exists M' \cdot M \overset{\hat{\ell}}{\Longrightarrow}_{\mathrm{p}} M' \ \wedge\ (M', N') \in R)$$

It is worth noting that weak refinement results in weak LTS bisimulation when restricted to MTS with no maybe transitions, and that strong MTS refinement implies weak refinement. Finally, as with strong refinement, a notion of implementation can be defined between MTSs and LTSs, the *weak semantics* of MTS can be defined in terms of sets of weak implementations, and it can be shown that $\preceq_{\mathrm{w}}$ implies inclusion of weak implementations.

Returning to our running example, recall model $\mathcal{C}$ described in Figure 3. If we hide actions *readList* and *showList* and then use weak refinement to compare it with the initial model $\mathcal{A}$, we can conclude that $\mathcal{C}$ is a refinement of $\mathcal{A}$ based upon the weak refinement relation $R = \{(0,0), (B,B), (M_1, M_1), (1_1, 1_1), \ldots, (1_{x_1}, 1_{x_1})\ ,\ (0, 2_{x_1})\}$. Thus, as expected, under weak semantics the more detailed model C is an adequate elaboration of the initial model $\mathcal{A}$.

One of the problems of weak MTS semantics is that it allows implementations that can be considered unintuitive: Consider the MTS $\mathcal{I}$ in Figure 4 which is an implementation of the original controller MTS $\mathcal{A}$ based on the weak implementation relation $R = \{(0,0), (B,B), (M_1, M_1), (1_1, 1_1), \ldots, (1_{x_1}, 1_{x_1})\}$.

Note that in $\mathcal{A}$ (Figure 1) the availability of $menu_n$ is yet to be defined, but if the system were to have this menu included we would expect all the functionalities associated with this menu to be reachable by the user. However in the implementation proposed above the user never has the possibility of selecting functionalities $func_1 \ldots func_{xn}$ after selecting $menu_n$. This breaks the intuition behind the notion of implementation. The implementation shown above is not satisfactory since it does not reflect the expected behaviour: if a menu is included, all its associated functionality will be available to users. This example shows that weak semantics does not seem to be adequate to support evolving software modelling since it accepts as valid refinements counter intuitive implementations. In subsequent sections we shall also show that weak semantics lacks some properties that relate refinement with action hiding, these properties are linked to some degree with the existence of such unintuitive implementations that weak semantics allows.

In summary, we have seen that although an observational semantics is required to support incremental elaboration of partial behaviour models, the observational semantics based on weak refinement not adequately fit with the intended meaning of MTS. In the next sections we show a semantics that not only resolves the case discussed above but that also provides a number of theoretical results that support the argument for a novel observational semantics for MTS.

4 Branching Semantics

In the previous section we analysed the shortcomings of strong and weak semantics as a foundation for characterising conformance and supporting model elaboration. Succinctly, strong semantics does not distinguish unobservable actions and hence does not support comparing models whose behaviour has been described to varying levels of detail. The latter allows implementations of partial models that contradict the intuition modellers may have of conformance. We now define a novel semantics for MTS that draws from desirable characteristics of both weak and strong semantics, in other words it is an observational semantics that captures the intuition that modellers might have of refinement. This novel semantics is based on LTS branching bisimulation.

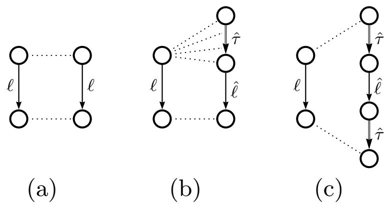

Fig. 5. Depiction of how a transition is simulated in bisimulation: (a) strong; (b) branching; (c) weak

$$\begin{tikzcd}
M \arrow[r, "\substack{\textit{branching}\\ \alpha\ \textit{extension}}"] \arrow[d, "\substack{\textit{branching}\\ \textit{impl}}"'] \arrow[dr, dotted, "\textit{branching}\ \alpha\ \textit{impl}"] & M' \arrow[d, "\substack{\textit{branching}\\ \textit{impl}}"] \\
I \arrow[r, "\substack{\textit{branching}\\ \alpha\ \textit{extension}}"'] & I'
\end{tikzcd}$$

Fig. 6. Informally, alphabet extension and branching implementations commute

Unlike strong bisimulation, branching bisimulation allows one LTS to simulate the occurrence of an ℓ transition in the other LTS by taking a number of τ transitions beforehand. Unlike weak bisimulation, branching bisimulation requires the intermediate states reached through τ transitions to fall within the equivalence relation. Figure 5 shows a graphical representation of how an ℓ transition is simulated in each of these three bisimulations. A branching implementation relation for MTS can be derived from LTS branching bisimulation in a similar manner as weak and strong implementation can be derived from weak and strong bisimulation.

Definition 9 (Branching Implementation Relation). *A* branching implementation relation *is a binary relation R from MTS to LTS such that whether $(M, I) \in R$ and $\ell \in Act_\tau$ the following holds:*

1. $(M \xrightarrow{\ell}_{\mathrm{r}} M') \implies (\exists\, I_0, \ldots, I_n, I') \cdot (I_0 = I \,\wedge\, I_i \xrightarrow{\tau} I_{i+1}\, \forall\, 0 \le i < n \,\wedge$
$I_n \xrightarrow{\hat{\ell}} I' \,\wedge\, (M', I') \in R \,\wedge\, (M, I_i) \in R\, \forall\, 0 \le i \le n)$
2. $(I \xrightarrow{\ell} I') \implies (\exists\, M_0, \ldots, M_n, M') \cdot (M_0 = M \wedge M_i \xrightarrow{\tau}_{\mathrm{p}} M_{i+1}\, \forall\, 0 \le i < n \,\wedge$
$M_n \xrightarrow{\hat{\ell}}_{\mathrm{p}} M' \,\wedge\, (M', I') \in R \,\wedge\, (M_i, I) \in R\, \forall\, 0 \le i \le n)$

Definition 10 (Branching Implementation). *Let M be an MTS and I be an LTS, we say that I is a* branching implementation *of M, $M \preceq_{\mathrm{b}} I$, if there* *exists a branching implementation relation R such as $(M, I) \in R$. We also define the set of implementations of M as $\mathcal{I}_b[M] = \{I \text{ LTS} \mid M \preceq_{\mathrm{b}} I\}$.*

As expected if this relation is restricted to LTS it coincides with branching equivalence. It can also be easily proved that if $M \preceq_{\mathrm{b}} I$ and $I \approx_{\mathrm{b}} I'$ then $M \preceq_{\mathrm{b}} I'$, and so this novel implementation relation is a sound extension of branching equivalence. It is worth mentioning that this new implementation relation does not accept as a valid implementation of model $\mathcal{A}$ depicted on Figure 1 the counter intuitive implementation shown on Figure 4.

Recalling that an MTS semantics is completely defined by stating which are valid implementations for a model, we define branching semantics based on the novel implementation relation instead of a refinement relation. An associated notion of refinement comes naturally as N is a refinement of M if all the implementations of N are implementations of M, as stated on definition 11.

Definition 11 (Branching Refinement). *Let M and N be MTSs, we say that N is a refinement of M, written $M \preceq_{\mathrm{b}} N$, iff $\mathcal{I}_b[M] \supseteq \mathcal{I}_b[N]$.*

Unlike refinement notions given by a simulation relation between MTSs this refinement notion is by definition complete. A co-inductive relation between MTS that implies branching implementation relation, mimicking Larsen's strong and weak refinement can easily be defined too. In the following section we will see how it is possible to demonstrate properties of this complete notion of refinement and to compare it against weak and strong refinement.

Definition 12 (Hiding). *Let $M = (S, L, \Delta^r, \Delta^p, s_0)$ be an MTS and $X \subseteq Act$. M with actions X hidden, denoted $M \backslash X$, is an MTS $(S, L \backslash X, \Delta^{r'}, \Delta^{p'}, s_0)$, where $\Delta^{r'} = \{(s, \ell, s') \mid \ell \notin X \wedge (s, \ell, s') \in \Delta^r\} \cup \{(s, \tau, s') \mid \ell \in X \wedge (s, \ell, s') \in \Delta^r\}$ and analogously for $\Delta^{p'}$. We use $M@X$ to denote $M \backslash (Act \backslash X)$.*

Branching refinement, similarly to weak refinement, does not allow for the comparison of models with different alphabets. However, we can do so by using the hiding operator, i.e. hiding the new labels of the extended alphabet. For example, given a model M and a model N, the latter with an alphabet that extends the alphabet of M, i.e. $\alpha M \subseteq \alpha N$, in order to assess whether N is a refinement of M we compute $M \preceq N@\alpha M$.

This operation gives a new refinement, therefore defining a new semantics for MTSs for which is possible to extend the alphabet of the models. In previous work [19,2] a similar extension has been applied to weak semantics, although this has been done implicitly without distinguishing between weak semantics and the extended alphabet semantics. However, since the set of implementations defined by branching implementation and the set obtained by applying this new refinement operator are different, they refer to two different semantics and we will make that distinction clear by formally defining this new semantics.

Definition 13 (Branching Alphabet Refinement). *An MTS N is a* branching alphabet refinement *of an MTS M, written $M \preceq_{\mathrm{ab}} N$, if $\alpha M \subseteq \alpha N$ and $M \preceq_{\mathrm{b}} N@\alpha M$.*

Note that this new semantics is an extension of branching semantics, as they behave in the same way when comparing models with identical alphabets. Similarly, we can define *Weak Alphabet Refinement* as an extension of weak refinement.

We now show that a sound relationship between branching implementation semantics and alphabet extension exists, but previously we define formally *equivalence* and *alphabet extension* for MTS.

Definition 14 (Equivalence). *Given a refinement for MTS, $\preceq$, we say that M and N are* equivalent, *written $M \approx N$, iff $M \preceq N$ and $N \preceq M$. We shall sometimes subindex $\approx$ to explicit the underlying refinement relation, e.g. $\approx_{\mathrm{b}}$ for branching refinement $\preceq_{\mathrm{b}}$.*

Definition 15 (Alphabet Extension). *Given an observational refinement for MTS, $\preceq$, we say that M' is an* alphabet extension *of M iff $M'@\alpha M \approx_{\mathrm{w}} M$.*

Theorem 1 (Branching semantics is sound w.r.t Alphabet Extension). *Let M be an MTS and I be an LTS such that I is a branching implementation of M, i.e. $M \preceq_{\mathrm{b}} I$. Given M' an MTS such that is a branching alphabet extension of M then there exists I' a branching alphabet extension of I such that $M' \preceq_{\mathrm{b}} I'$.*

Intuitively, if a model M is extended into a model M' then all implementations of M can be extended to be an implementation of M'. Figure 6 provides an intuition of Theorem 1. We say, informally, that the diagram commutes, meaning that it is possible to obtain the same result by taking an implementation of M and then extending the alphabet of that implementation; or by extending the alphabet of M and then taking an implementation of that model.

From an engineering perspective this result implies that whatever implementation we have in mind for a given partial model, refining the alphabet of the partial model will not rule out that implementation: extending the original implementation to make it an implementation of the new model is possible.

It is important to note that it is not possible to formulate a similar soundness result as the one above under weak semantics:

Remark 1 (Weak semantics is not sound w.r.t Alphabet Extension). Let M and M' be MTSs such that M' is a weak alphabet extension of M. It is not the case that for all LTS I such that $M \preceq_{\mathrm{w}} I$ then there exists I' such that $M' \preceq_{\mathrm{w}} I'$ and I' is weak alphabet extension of I.

Proof. Consider the example described in the previous section. Assume we extend model $\mathcal{A}$ given in Figure 4 to produce $\mathcal{A}'$ by extending its alphabet with the label *timeout*, and replacing τ transitions from M_i to state B with a *timeout* transition. It would be reasonable to expect that model I could be extended with *timeout* into a I' to obtain an implementation of $\mathcal{A}'$. However, this is not possible. If we analyse this in further detail, we can see that we would need I' to be able to perform a *timeout* after $menu_n$ and before reaching state B. Hence, I' would have a new state in between $menu_n$ and *timeout.* This leads to one of two options, either the new state does not simulate the require behaviour of M_n because it does not have transitions $func_1 ... func_{x_n}$, and therefore I' could not be an implementation of A'; or it does have those transitions and refines state M_n, but in this case $I'@\alpha I$ would not be equivalent to I since I does not have any of the functionalities available after $menu_n$ and therefore I' could not be an alphabet extension of I.

Summarising, in this section we have defined a new observational semantics for MTS that preserves the branching structure and resolves the unintuitive example provided in the motivation section. Furthermore, we have formally defined an extension of this semantics that supports not only the elaboration of model behaviour but also the extension of their alphabets, laying the foundations for a sound elaboration process where the level of the detail of the models can be increased over time. We have also shown that extending the alphabet of a partial behaviour model is a sound operation with respect branching semantics, while it is not for weak semantics.

5 Consistency

In this section we discuss the notion of consistency which is central to MTS semantics. We provide a complete characterization of consistency under branching semantics (result unavailable for weak semantics) and show that, unlike in weak semantics, consistency is preserved by hiding non-shared actions.

In order to support elaboration of partial behaviour models, a number of operations over MTS have been studied. Most notably, Larsen defined two co-inductive relations [16,11] which allow checking efficiently if there is a subset relation between the implementations of two MTS. This allows elaborating an MTS and checking if the new MTS effectively only "adds information", i.e. reduces acceptable implementations, to the first MTS. Another useful operation is that of merge [7,19], which attempts to produce an MTS that characterises the common implementations of two given MTS. This operation which is a form of conjunction [15] supports composing partial descriptions provided by different modellers possibly with different scopes or viewpoints of the same system. Finally, checking if two partial descriptions are consisent, in other words that there is at least one implementation that conforms to both descriptions is a pre-condition for merging and a usefull operation in its own right for understanding the relation between different partial descriptions.

In this section, we analyse the notion of consistency under branching semantics and also compare with weak semantics. The study of a co-inductive refinement relation, which can be easily formulated, and merge under branching semantics is left out of this paper due to space restrictions and the fact that within consistency lie some key results that distinguish branching from weak semantics. We start with a formal definition of consistency.

Definition 16 (Consistency). *Two MTSs M and N are* consistent *if there exists an MTS P such that P is a common refinement of M and N.*

The problem of characterising consistency has been solved for strong semantics in [7] where a sufficient and necessary condition for determining if there exist a common strong refinement for two models is presented. We now define a new relation, *branching alphabet consistency relation*, and show that it characterises branching alphabet consistency.

Definition 17 (Branching Alphabet Consistency Relation). *A* branching alphabet consistency relation *is a binary relation $C \subseteq \delta \times \delta$, such that the following conditions hold for all $(M, N) \in C$:*

1. $(M \xrightarrow{\ell}_{\mathrm{r}} M') \implies (\exists N_0, \ldots, N_n, N') \cdot ((N_i \xrightarrow{v}_{\mathrm{p}} N_{i+1} \wedge v \notin \alpha M)\ \forall 0 \le i < n \wedge N_0 = N \wedge N_n \xrightarrow{\hat{\ell}}_{\mathrm{p}} N' \wedge (M, N_i) \in C\ \forall 0 \le i \le n \wedge (M', N') \in C$
2. $(N \xrightarrow{\ell}_{\mathrm{r}} N') \implies (\exists M_0, \ldots, M_n, M') \cdot ((M_i \xrightarrow{v}_{\mathrm{p}} M_{i+1} \wedge v \notin \alpha N)\ \forall 0 \le i < n \wedge M_0 = M \wedge M_n \xrightarrow{\hat{\ell}}_{\mathrm{p}} M' \wedge (M_i, N) \in C\ \forall 0 \le i \le n \wedge (M', N') \in C$

Intuitively, this relation requires that one model provides as possible behaviour at least all the required behaviour of the other, and vice versa.

The branching consistency relation defined above characterises branching alphabet consistency in the sense that there is a branching alphabet consistency relation between two MTS if and only if there exists an LTS that is a branching alphabet implementation of the two MTS.

Theorem 2 (Characterisation of Branching Alphabet Consistency). *MTSs M and N are branching alphabet consistent if and only if there exists a branching alphabet consistency relation C_{MN} such that (M, N) is in C_{MN}.*

Note that the Branching Alphabet Consistency Relation is equivalent to branching bisimulation when restricted to LTSs with the same alphabet This result is as expected, since an LTS is an MTS that characterises only one implementation, itself. Hence, it can only be consistent with any LTS that is equivalent to it; equivalence which in this case is that of LTS branching bisimulation.

Similar results do not exist for weak semantics. In [2] a first attempt to characterise weak consistency was published, however the definition has some problems. An improvement of the weak consistency relation in [2] is:

Definition 18. (Weak Alphabet Consistency Relation) *A* weak alphabet consistency relation *is a binary relation $C \subseteq \wp \times \wp$, such that the following conditions hold for all $(M, N) \in C$:*

1. $(M \xrightarrow{\ell}_{\mathrm{r}} M') \implies (\exists N') \cdot (N \overset{v\hat{\ell}w}{\Longrightarrow}_{\mathrm{p}} N' \wedge v, w \in (\alpha N \setminus \alpha M)^* \wedge (M', N') \in C))$
2. $(N \xrightarrow{\ell}_{\mathrm{r}} N') \implies (\exists M') \cdot (M \overset{v\hat{\ell}w}{\Longrightarrow}_{\mathrm{p}} M' \wedge v, w \in (\alpha M \setminus \alpha N)^* \wedge (M', N') \in C))$

Theorem 3 (Characterisation of Weak Consistency). *Two MTSs M and N, such that $\alpha M = \alpha N$, are weak consistent if and only if there exists a weak alphabet consistency relation C_{MN} such that (M, N) is contained in C_{MN}.*

The weak alphabet consistency relation restricted to models with the same alphabet characterises weak consistency, this can be easily proved using the Theorem 1 presented in [7]. However, it does not characterise weak alphabet consistency. Figure 7 shows a counter example, models M and N with alphabets

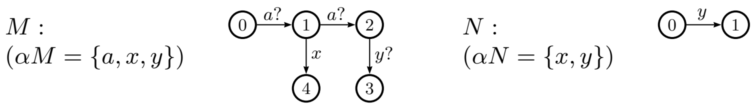

Fig. 7. Counter example for weak alphabet consistency characterisation

$\alpha M = \{a, x, y\}$ and $\alpha M = \{x, y\}$ are not consistent but $C_{MN} = \{(0, 0), (3, 1)\}$ is a valid relation. Definition 18 can be made more restrictive giving it a branching feel in line with [2] obtaining a relation that is a sufficient but not a necessary condition for weak alphabet consistency. This relation is out of the scope of this paper and for space limitation is not included.

In the same way Theorem 1 relates refinement with alphabet extension, it is interesting and relevant to analyse the relation between consistency and alphabet extension. Here we also find that the expected results hold for branching semantics but do not for weak semantics.

The following theorem establishes that models are branching alphabet consistent if and only if they are branching consistent over their common alphabet.

Theorem 4. *Let M and N be MTSs, and $A = \alpha M \cap \alpha N$ be the common alphabet of M and N. $M@A$ and $N@A$ are branching consistent iff M and N are branching alphabet consistent.*

From an engineering point of view this theorem expresses the fact that in order to assess whether two models are consistent it is sufficient to evaluate whether they are consistent in their common alphabet. On the other hand, it tells us that given two consistent models with the same alphabet it is possible to elaborate those models independently, extending their alphabets over different labels, knowing that the models will always remain consistent. This is a useful feature, especially when comparing two models taken from different viewpoints of the system, and for which there is a requirement to increase the level of detail with regards to different aspects. Interestingly, the natural candidate for weak alphabet consistency relation does not satisfy the left-to-right implication of the above theorem. In other words that if two models are weak consistent, extending them over new labels does not guarantee that they will remain consistent.

A related result, that in a way is more general than Theorem 4 is shown below. Note that the converse Theorem 5 is not generally true, but in the particular case of Theorem 4 the converse is also true and it can be trivially proved.

Theorem 5. *Let M' and N' be MTSs, and $A = \alpha M' \cap \alpha N'$ be the common alphabet of M' and N'. If there exist M and N MTSs such as $M'@A \preceq_{\mathrm{ab}} M$, $N'@A \preceq_{\mathrm{ab}} N$ and M and N are branching alphabet consistent then M' and N' are branching alphabet consistent.*

In summary, have provided a complete characterization for consistency under branching semantics and shown that it has the expected properties when considered in the context of alphabet extension. These results do not exist for weak refinement of MTS.

6 Related Work

Various authors have contributed to the study of MTS and other partial behaviour modelling formalisms. Our definition of Modal Transition Systems differs from the original [16] in that MTS can have different communication alphabets. Expliciting the communication alphabet allows scoping models and capturing the fact that components control and monitor a subset of all events [12].

Related work regarding MTS refinement and simulation has been discussed extensively throughout the paper. Our notions of branching refinement and branching implementation are heavily inspired on that of branching bisimulation, although as shown, the extension of branching bisimulation from LTS to MTS cannot be done straightforwardly. Numerous other refinement notions exist, both for LTS (such as trace, failures [18], and testing [4] refinement) and for other state-based modelling formalisms such as kripke structures. We have also compared extensively the notion of refinement we propose with respect to strong [16] and weak refinement [16] over MTS. Regarding consistency, as mentioned previously, a characterization of consistency under strong semantics has been developed previously [7], while up to now no characterization of consistency for weak nor weak alphabet semantics had been provided (the definition in [2] fails to do so completely). In [8] we sketch the idea of a branching semantics however the theoretical results presented in this paper are novel.

Numerous extensions and variants of MTS exist such as Mixed Transition Systems [5] and disjunctive modal transition systems [14]. The semantics we propose could be studied for these formalisms too. We believe that existing weak and strong refinement notions in these settings will suffer from the same shortcomings as in MTSs. A slightly different approach to modelling unknown behaviour is taken in [20,17]. In [20] Partial Labelled Transition Systems, each state is associated with a set of actions that are explicitly proscribed from happening. Extended Transition Systems [17] also associate a set of actions with each state, but in this case it models the actions for which the state has been fully described. The relation between these models and MTS, and in particular, our notion of refinement has yet to be studied.

In [1] a study of the complexity of different decision problems for MTS and Mixed transition systems is presented. In particular it is shown that thorough refinement for strong and weak semantics is PSPACE-hard, considering that branching alphabet refinement is between these two is expected to have the same complexity but further study is necessary.

7 Conclusions and Future Work

In this paper we have analysed the limitations of existing semantics for MTS and presented a new observational semantics, called branching semantics, based on the notion of branching equivalence. Furthermore, we have shown how this new semantics does not allow for the counter-intuitive implementations permitted by weak semantics. Moreover, in order to allow for the elaboration of models' alphabets, we have distinguished branching semantics from branching alphabet semantics. Lastly, we have shown how branching alphabet semantics presents a series of desirable properties that are not valid for weak semantics, making it a more adequate option for model elaboration.

In future work we aim to study the problem of merging under alphabet branching semantics.

References

1. Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wasowski, A.: Complexity of decision problems for mixed and modal specifications. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 112–126. Springer, Heidelberg (2008)
2. Brunet, G.: A Characterization of Merging Partial Behavioural Models. Master's thesis, Univ. of Toronto (January 2006)
3. Brunet, G., Chechik, M., Uchitel, S.: Properties of behavioural model merging. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 98–114. Springer, Heidelberg (2006)
4. Cleaveland, R., Hennessy, M.: Testing equivalence as a bisimulation equivalence. Formal Asp. Comput. 5(1), 1–20 (1993)
5. Dams, D.: Abstract Interpretation and Partition Refinement for Model Checking. PhD thesis, Eindhoven University of Technology, The Netherlands (July 1996)
6. Fischbein, D., Uchitel, S.: Behavioural model elaboration using mts. In: "Copenhagen" Meeting on Modal Transition Systems (2007)
7. Fischbein, D., Uchitel, S.: On correct and complete strong merging of partial behaviour models. In: SIGSOFT 2008/FSE-16, pp. 297–307. ACM Press, New York (2008)
8. Fischbein, D., Uchitel, S., Braberman, V.: A foundation for behavioural conformance in software product line architectures. In: ROSATEA (2006)
9. van Glabbeek, R.: What is branching time semantics and why to use it? In: Nielsen, M. (ed.) The Concurrency Column, pp. 190–198 (1994); Bulletin of the EATCS 53
10. Huth, M.: Refinement is complete for implementations. Formal Asp. Comput. 17(2), 113–137 (2005)
11. Hüttel, H., Larsen, K.G.: The use of static constructs in a modal process logic. In: Logic at Botik, pp. 163–180 (1989)
12. Jackson, M.: Software requirements & specifications: a lexicon of practice, principles and prejudices. ACM Press/Addison-Wesley Publishing Co. (1995)
13. Keller, R.M.: Formal verification of parallel programs. Commun. ACM (1976)
14. Larsen, K., Xinxin, L.: Equation Solving Using Modal Transition Systems. In: 5th Annual IEEE Symposium on Logic in Computer Science, pp. 108–117 (1990)
15. Larsen, K.G., Steffen, B., Weise, C.: A constraint oriented proof methodology based on modal transition systems. In: Brinksma, E., Steffen, B., Cleaveland, W.R., Larsen, K.G., Margaria, T. (eds.) TACAS 1995. LNCS, vol. 1019. Springer, Heidelberg (1995)
16. Larsen, K.G., Thomsen, B.: A modal process logic. In: LICS (1988)

17. Milner, R.: A modal characterisation of observable machine-behaviour. In: Astesiano, E., Böhm, C. (eds.) CAAP 1981. LNCS, vol. 112, pp. 25–34. Springer, Heidelberg (1981)
18. Schneider, S., Schneider, S.A.: Concurrent and Real Time Systems: The CSP Approach. John Wiley & Sons, Inc., New York (1999)
19. Uchitel, S., Chechik, M.: Merging partial behavioural models. In: Taylor, R.N., Dwyer, M.B. (eds.) SIGSOFT FSE, pp. 43–52. ACM Press, New York (2004)
20. Uchitel, S., Kramer, J., Magee, J.: Behaviour Model Elaboration using Partial Labelled Transition Systems. In: ESEC/FSE 2003, pp. 19–27 (2003)
21. van Gabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. J. ACM 43(3), 555–600 (1996)