

Branching Semantics for Modal Transition Systems

Tesista

Dario Fischbein
dfischbe(at)dc.uba.ar
L.U. 472/99

Director

Sebastian Uchitel
s.uchitel(at)doc.ic.ac.uk
(Imperial College London)

Co-director

Victor Braberman
vbraber(at)dc.uba.ar

Abril 2006

Abstract

Modal Transition Systems (MTS) are a formalism that allow for partial descriptions of a system's behaviour. These models characterise the set of implementations that satisfy the partial knowledge available and facilitate the analysis of properties over this set. Given a model, the set of implementations it defines depends on the semantics used to interpret it. In this thesis we analyse the existing MTS semantics concluding they are not adequate for incrementally evolving a model from a software engineering perspective. We discuss the required characteristics for a semantics to be suitable and subsequently give a formal definition for a new semantics that has these characteristics. Finally, we present a software tool that we have developed to verify whether an implementation conforms to a partial model according to each of the studied semantics, i.e. if it is included in the set of implementations given by each semantics for that partial model.

Contents

Abstract	1
Contents	3
Introduction	6
1 Background	7
1.1 Equivalences	7
1.2 Refinements	11
1.3 Modal Transition Systems	13
2 Analysis of Existing Semantics	15
2.1 Case Study	15
2.2 Strong Semantics	17
2.3 Weak Semantics	20
3 New Semantics	23
3.1 Exploration	23
3.2 Definition	29

<i>CONTENTS</i>	3
4 Tool Support	33
4.1 Fix Point Algorithm	34
4.2 Implementation	36
4.2.1 FSP	37
4.2.2 User Guide	39
5 Validation	42
6 Conclusions	46
6.1 Summary of Contributions	46
6.2 Related Work	47
6.3 Future work	48
Bibliography	49

Introduction

The requirements and design of software systems are amenable to analysis through the construction of behaviour models, that is, formal operational descriptions of the intended system behaviour. This corresponds to the traditional engineering approach to construction of complex systems. The major advantage of using models is that they can be studied to increase confidence in the adequacy of the product to be built. In particular, behaviour models used to describe software systems can be analysed and mechanically checked for properties in order to detect design errors early in the development process and allow cheaper fixes.

Although behaviour modelling and analysis have been shown to be successful in uncovering subtle design errors [1], the adoption of such technologies by practitioners has been slow. This is in part due to a mismatch between most widely adopted software development techniques and a fundamental characteristic of traditional behaviour models.

On one side, as part of the essence of widely used iterative and incremental software development processes, the available system descriptions tend to be of a partial nature leaving some aspects of the desired behaviour undefined until a more advanced stage of the process is reached. Consequently, when the advantages of constructing models are more rewarding the complete system description is not available.

However, semantics of traditional behaviour models assume a complete description of the system behaviour up to some level of abstraction, and hence cannot support reasoning in the presence of partial behavioural information. To support such reasoning, a behaviour model should allow currently unknown as-

pects of system behaviour to be modelled, moreover it should provide a notion of elaboration of partial descriptions into more comprehensive ones.

Operational models that allow distinguishing unknown aspects of system behaviour are referred to as *partial behavioural models*. A particular kind of widely studied partial behavioural models are Modal Transition Systems (MTS), an extension of Label Transition Systems (LTS).

With these type of models we can describe what is already known of the desired behaviour of the system at an early stage of the software development process, and also analyse it in spite of not having complete knowledge on the expected system behaviour. Further on, once all behavioural information is available, we will need a way to ensure that the final and complete model constructed actually conforms to the partial models initially developed. If this is the case we say that the total model is an implementation of the partial model. If the notion of conformance can be shown formally to preserve certain properties, then all analysis previously done over the partial models will also be valid for the implementation.

Refinement is a generalization of the notion of implementation that allows us to establish when one partial model conforms to another one. Intuitively, model N is a refinement of model M if and only if the set of implementations of model N is a subset of the implementations of M. With this notion we can start with an immature model and gradually and repeatedly evolve it while new knowledge is gathered until the complete system description is reached.

There exist two different semantics over MTS given by two refinements introduced by Larsen et al. [2, 3]. As part of this work we analyse from an incremental software modelling point of view the applicability of the implementation notions derived from Larsen's refinements. Based on our analysis we conclude that from this perspective these implementation notions are not adequate.

To address the limitations of existing semantics, we define a new semantics for MTS given by a novel notion of implementation. This semantics has a correspondence with branching bisimulation[4] over LTS in a similar way Larsen's semantics informally correspond to strong and weak bisimulation over LTS. In

addition, we show that the *natural* generalization of branching bisimulation does not lead to an appropriate notion of refinement over MTS. Hence, in this work we provide a declarative definition of branching refinement over MTS but leave an operationally definition, in the style of Larsen [2, 3] for future work. Finally, we present a tool we have developed that checks the implementation relation between any two models using strong, weak, or the new semantics. We have used this tool to validate the examples presented in the work.

The rest of this thesis is organized as follows. In Chapter 1, we provide the background for the rest of the work introducing LTS, some semantics for them and finally defining MTS. In Chapter 2, we present and analyse the existing semantics for MTS. In Chapter 3, we define a novel semantics for MTS. Chapter 4 gives an algorithm to verify the implementation relation between models and introduce a tool that has been developed to support the verification of the different relations discussed in this work. In Chapter 5, we validate the novel semantics against a set of analysed examples. Finally, in Chapter 6 we conclude with a summary of contribution, an analysis of related work and future directions.

Chapter 1

Background

In this chapter we introduce LTS, three standard equivalence relations and two standard refinements on them, and also MTS. We provide formal definitions, some intuitions and examples, and fix notation.

1.1 Equivalences

Definition 1.1.1 (Labelled Transition Systems). *Let $States$ be a universal set of states, Act be a universal set of observable action labels, and let $Act_\tau = Act \cup \{\tau\}$. A labelled transition system (LTS) is a tuple $P = (S, L, \Delta, s_0)$, where $S \subseteq States$ is a finite set of states, $L \subseteq Act_\tau$ is a set of labels, $\Delta \subseteq (S \times L \times S)$ is a transition relation between states, and $s_0 \in S$ is the initial state. We use $\alpha P = L \setminus \{\tau\}$ to denote the communicating alphabet of P .*

Figure 1.1 shows a graphical representation of two LTSs. 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 $P \xRightarrow{\ell} P'$ to denote $P(\xrightarrow{-\tau})^* \xrightarrow{\ell} (\xrightarrow{-\tau})^* P'$. Let $w = w_1, \dots, w_k$ be a word over Act_τ . Then $P \xrightarrow{w} P'$ means that there exist P_0, \dots, P_k such that $P = P_0$, $P' = P_k$, and $P_i \xrightarrow{w_{i+1}} P_{i+1}$ for $0 \leq i < k$. We write $P \xrightarrow{w}$ to mean $\exists P' \cdot P \xrightarrow{w} P'$. Finally, we extend \implies to words in the same

way as we do for \longrightarrow .

Definition 1.1.2 (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. $(P \xrightarrow{\ell} P') \Rightarrow (\exists Q' \cdot Q \xrightarrow{\ell} Q' \wedge (P', Q') \in R)$
2. $(Q \xrightarrow{\ell} Q') \Rightarrow (\exists P' \cdot P \xrightarrow{\ell} P' \wedge (P', Q') \in R)$

Informally, two models are strong equivalent if their initial states are strong equivalent, and two states are strong equivalent if, whenever one action can be executed in one of them leading to a state B , the other can execute the same action reaching a state B' , where B' is again equivalent to B . This equivalence does not distinguish τ transitions as special or unobservable actions. A property of this equivalence is that it respects the branching structure of processes[5].

Consider the LTS shown in Figure 1.1. These two models are an example of strong equivalent models, and the bisimulation relation between them is $R = \{(a_0, b_0), (a_1, b_1), (a_2, b_2), (a_0, b_3), (a_1, b_4), (a_2, b_5)\}$. On the other hand Figure 1.2 shows an example of two LTS that are not strong equivalent. There not exist a strong bisimulation for these models because state 1 of the model in 1.2(a) cannot be related with any state of the model in 1.2(b).

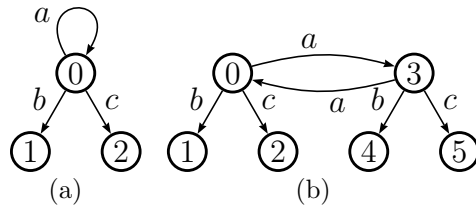


Figure 1.1:

Definition 1.1.3 (Weak Bisimulation Equivalence). *Let \wp be the universe of all LTS, and $P, Q \in \wp$. P and Q are weak bisimulation equivalent, written $P \approx Q$, 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 $\ell \in Act_\tau$:*

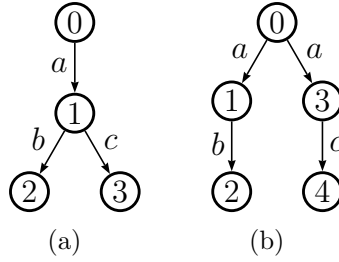


Figure 1.2:

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

This equivalence compares the observational behaviour of models ignoring silent actions (τ -transitions). Some authors call this equivalence *observational equivalence*, but we are going to use this expression to refer to any equivalence that considers τ -transitions as silent actions. Weak bisimulation equivalence is coarser than strong equivalence and does not preserve the branching structure of processes as it is shown in [4].

Figure 1.3 shown an example of two models that are not strong equivalent but weak equivalent. The weak bisimulation relation between them is $R = \{(a_0, b_0), (a_1, b_1), (a_2, b_1), (a_3, b_3)\}$.

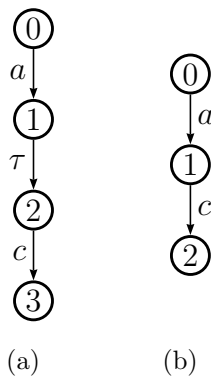


Figure 1.3:

Definition 1.1.4 (Branching Bisimulation Equivalence). *Let \wp be the universe of all LTS, and $P, Q \in \wp$. P and Q are branching bisimulation equivalent, written*

$P \approx_b Q$, 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 $\ell \in \text{Act}_\tau$:

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

Lemma 1.1.5 (Stuttering Lemma). *Let R be the largest branching bisimulation between P and Q . If $r \xrightarrow{\tau} r_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} r_m \xrightarrow{\tau} r'$ ($m \geq 0$) is a path such that $(r, s), (r', s) \in R$ then $(r_i, s) \in R \forall i \leq m$.*

Branching bisimulation equivalence is an observational equivalence coarser than strong equivalence and finer than weak bisimulation equivalence. This equivalence is the coarsest equivalence that preserves the branching structure of processes[5]. The Stuttering Lemma will have an important role in the results we present in section 3.2.

Figure 1.4 shows an example of two models that are weak equivalent but not branch equivalent. If we make a comparative analysis of models (a) and (b) we can see that while from the initial state in model (a) we can take transition a in only one way, in model (b) there are two different possibilities to take this transition from state 0. One of those two possibilities leads to a state from where both b_1 and b_2 transitions can be taken. This is the same that happens if we take transition a in model (a). However, if from the initial state in model (b) we take

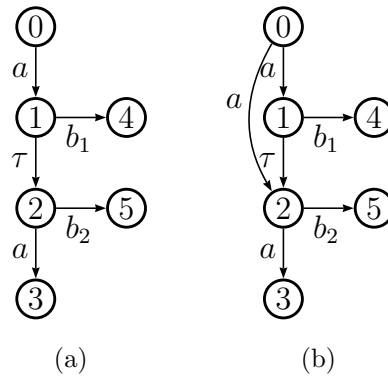


Figure 1.4:

transition a leading to state 2 then the possibility of taking b_1 is discarded before having the chance of taking it, which is never the case in model (a). Therefore, we can conclude that these two models do not have the same branching structure.

1.2 Refinements

In the previous section we present a series of equivalences over LTS. While all these equivalences determine whether two models have or not the same behaviour, they differ in the criteria used to interpret the behaviour given by a model.

However, in the context of evolving a software model we need to be able to add further information to the model while more knowledge regarding the system is acquired. This implies that we do not only need to be able to assess if two models are equivalent or not but to define if a model with more information actually refines a previous one. In order to do so, we will need to find a suitable semantics that establishes an order between models.

In this section we present two different refinement notions over LTS and we analyse why they do not allow us to evolve a model according to our expectations.

Definition 1.2.1 (Strong Simulation [6]). *Let \wp be the universe of all LTS, and $P, Q \in \wp$. Q simulates P , written $P \sqsubseteq_s Q$, if $\alpha P = \alpha Q$ and (P, Q) is contained in some simulation relation $R \subseteq \wp \times \wp$ for which the following holds for all $\ell \in Act_\tau$:*

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

While Strong Bisimulation defines an equivalence, Strong Simulation defines a partial order over LTS. Note that $P \sim Q \Rightarrow P \sqsubseteq_s Q \wedge Q \sqsubseteq_s P$.

A standard refinement notion between LTS consists in considering that P refines Q if P simulates Q . i.e. $Q \sqsubseteq_s P$. One of the properties that characterize this refinement is that it reduces the degree of non determinism. With this refinement we know that any behaviour of Q is a valid behaviour of P . However, using this semantics it is not possible to ensure that a forbidden behaviour for

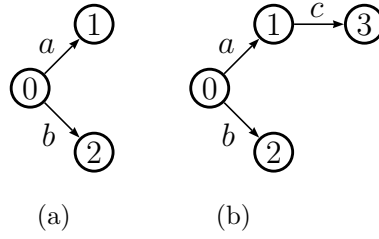


Figure 1.5:

the system which is captured in Q will be preserved in the refined model P . For example, the model shown in figure 1.5(b) refines the one shown in figure 1.5(a) and it can be easily seen that model 1.5(b) has the possibility of taking transition c after taking a in spite of being forbidden in model 1.5(a). Moreover, a model consisting in a state with a self-transition for every element of the alphabet is considered by this semantics a refinement of any other model.

On the other hand, if we consider that P refines Q if Q simulates P , i.e. $P \sqsubseteq_s Q$. In this case we know that any behaviour of P is a valid behaviour of Q , but not the other way around. Therefore, if Q determines that certain behaviour is forbidden in a system we know that it will also be forbidden by P since any behaviour in P can be simulated by Q . However, using this semantics it is not possible to ensure that a desired behaviour for the system which is depicted in Q will be preserved in the refined model P . For example, the model shown in figure 1.5(a) refines the one shown in figure 1.5(b) and it can be easily seen that model 1.5(a) loses the possibility of taking transition c after taking transition a . Moreover, a model consisting in a state with no transitions is, considered by this semantics, a refinement of any other model.

Definition 1.2.2 (Traces). *A trace is a sequence $w \in Act^*$. Given P a LTS, the set of traces of P is defined as follows:*

$$TRACES(P) = \{ w \in Act^* \mid P \xrightarrow{w} \}$$

Definition 1.2.3 (Trace Refinement). *Let P and Q be LTSs. We say that Q is a trace refinement of P , written $P \sqsubseteq_{tr} Q$, iff $TRACES(P) \supseteq TRACES(Q)$.*

Suppose an LTS model P describes the knowledge we have at a certain stage

of the development process. Then, according to trace refinement semantics, an LTS model Q will be a valid implementation of P iff the set of traces of Q is a subset of the traces of P . Therefore, P determines the set of all possible traces for the system but cannot guarantee any of them. This means that while P describes the “maximum” behaviour possible for the implementations it cannot assure any required behaviour will be preserved. For example, according to this semantics, a model consisting of only one state and no transitions is a possible implementation of any system. Moreover, the definition of trace refinement is such that the inclusion of any trace is independent from the inclusion of other traces (except for its prefixes). Consequently, using this semantics it is not possible to describe a requirement such as ‘if the implementation has this specific behaviour then it must have this other behaviour’.

Bearing in mind our aim of supporting the evolution of behavioural models having partial information as a starting point and enriching the model while more requirements are gathered, the refinements presented in this section have the following limitation: they consider the less refined model to completely describe either the maximum or the minimum allowed behaviour for the system. In the first case all possible behaviour is specified whereas in the second case the model describes all the required behaviour for the system. This means only one bound is specified, i.e. either the lower or upper bound, and hence the other bound remains open. In fact, this limitation is not due to the analysed refinement notions in themselves, failures [7], testing [8] refinements have similar problems, but to an intrinsic characteristic of LTS models. Specifically, using these models it is not possible to identify which aspects of the system have already been defined and which still has to be refined. This is true both for the expected and the forbidden behaviour. For this reason we will consider MTS since they allow us to explicitly specify which behaviour is required, possible or forbidden.

1.3 Modal Transition Systems

Definition 1.3.1 (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 rep-*

representing required transitions of the system and (S, L, Δ^p, s_0) is an LTS representing possible (but not necessarily required) transitions of the system. We use $\alpha M = L \setminus \{\tau\}$ to denote the communicating alphabet of M .

Figure 2.1 shows a graphical representation of a MTS. Transition labels that have a question mark are those in $\Delta^p - \Delta^r$. We refer to these transitions as “maybe” transitions, to distinguish them from required ones (those in Δ^r). 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 $(s_0, \ell, s'_0) \in \Delta^p$. Similarly, for $\gamma \in \{r, p\}$ we write $M \xrightarrow{\hat{\ell}}_{\gamma} M'$ to denote that either $M \xrightarrow{\ell}_{\gamma} M'$ or $\ell = \tau$ and $P = P'$ are true, and we use $P \xRightarrow{\ell}_{\gamma} P'$ to denote $P(\xrightarrow{\tau}_{\gamma})^* \xrightarrow{\ell}_{\gamma} (\xrightarrow{\tau}_r)^* P'$.

Note that LTS are a special type of MTS that do not have maybe transitions.

Chapter 2

Analysis of Existing Semantics

In this section we analyse the adequacy of MTS refinement as introduced by Larsen for software development, especially in iterative and incremental software development processes. We present a case study upon which we analyse the use of the implementation notions given by strong and weak MTS refinements.

2.1 Case Study

The case study that we will analyse consists of a model of the control software for an electronic device at an early stage of the development process. This device offers different functions grouped into several menu items. The general behaviour of the system is basically as follows: the user selects the desired menu item and the system offers the functions of this menu entry. If the user does not choose any function after an elapsed time, the system makes a beep and returns to the initial state. At this stage of the development process some menu items and their associated functionalities have been discovered, but the stakeholders are not sure whether some of the menu entries described will be adequate or not for the final product. According to this situation, the MTS that models the system with the detail and knowledge that we have up to now is shown in Figure 2.1.

What is being modelled at this stage is the way the user can access the

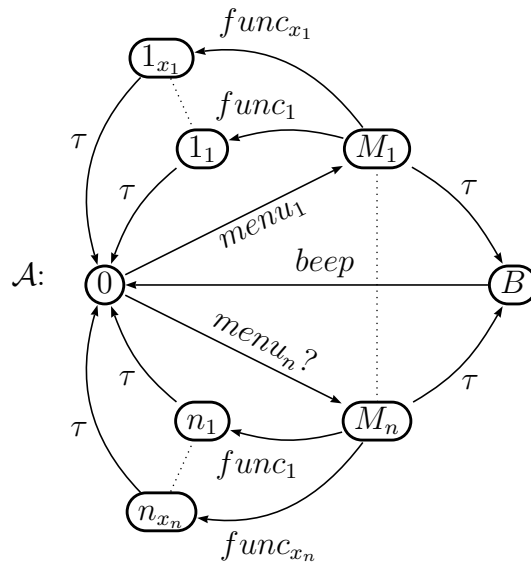


Figure 2.1: A graphical representation of the MTS that models the system with the detail and knowledge gathered at the first stage. The labels of the states have no meaning and are used for reference only. The initial state of the system is denoted with label 0.

functionalities of the equipment. How these functionalities work is below the level of abstraction of this model and are consequently represented with τ transitions.

From the initial state there are n transitions labelled $menu_1$ to $menu_n$ each one representing the selection of one menu entry 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.

When an M_i state is reached, meaning that the user has selected the menu i , a set of transitions $func_1$ to $func_{x_i}$ become available. These transitions represent the access to the functionalities that can be triggered from this menu item. If the user triggers one of these, the system will carry out the associated task and after finishing that task it will return to the initial state. As we have already mentioned, since the details of how these tasks are performed is below the level of abstraction of the model, they have been represented with τ transitions. If the user has not selected any functionality after an elapsed time then an internal timeout occurs, making the system leave the M_i state and return to the initial state by doing a *beep*. This time out is an internal event and therefore not visible

to the user, so again it has been modelled with a τ transition.

2.2 Strong Semantics

When considering LTS, *strong semantics* refers to the semantics given by strong bisimulation. One of the particularities of this semantics is that it lacks a notion of unobservable or internal action, i.e. τ -labelled transitions. Larsen has extended this semantics over MTS [2].

Strong refinement 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 2.2.1 (Strong Refinement). *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. $(M \xrightarrow{\ell}_r M') \Rightarrow (\exists N' \cdot N \xrightarrow{\ell}_r N' \wedge (M', N') \in R)$
2. $(N \xrightarrow{\ell}_p N') \Rightarrow (\exists M' \cdot M \xrightarrow{\ell}_p M' \wedge (M', N') \in R)$

Note that the second condition guarantees that if N has a required transition, M has a maybe or a required transition, whereas if N has a maybe transition, then M has a maybe transition – otherwise, the first condition is violated. Another interesting thing to note is how similar is this definition to strong bisimulation Def. 1.1.2.

Consider the MTS shown in Figure 2.1. If modellers decide to exclude $menu_n$ then the model that would represent that decision is the one shown in Fig-

ure 2.2. 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 $R = \{(0, 0), (B, B), (M_1, M_1), (1_1, 1_1), \dots, (1_{x_1}, 1_{x_1})\}$.

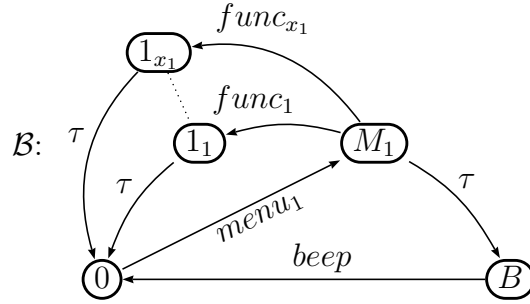


Figure 2.2: A possible evolution of the initial model where only $menu_1$ is available to the user.

Note that because the MTS \mathcal{B} in Figure 2.2 has no maybe transitions can be considered an LTS and we say that it is an implementation of the model in Figure 2.1.

Definition 2.2.2 ((strong) implementation). *We say that an LTS $I = (S_I, L_I, \Delta_I, i_0)$ is a (strong) implementation of an MTS $M = (S_M, L_M, \Delta_M^r, \Delta_M^p, m_0)$, 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 $\mathcal{I}[M] = \{I \text{ LTS} \mid M \preceq I\}$.*

Considering that strong refinement is transitive [2] it is straightforward to proof that $M \preceq M'$ implies $\mathcal{I}[M] \supseteq \mathcal{I}[M']$. Hence when a model is evolved into a more refined one no new possible implementations for the system will be added, in fact when a model is enriched with more requirements the set of possible implementations can only be reduced. Another important result is that the reciprocal of the previous property is also valid, $\mathcal{I}[M] \supseteq \mathcal{I}[M'] \Rightarrow M \preceq M'$ [9]. A consequence of this is that when a modeller creates a new model to reflect additional requirements that were gathered we can be sure that this model will be accepted as a valid evolution of the previous model iff the set of implementations of the new model is a subset of the implementations of the previous one.

A remarkable property of this refinement is that it preserves the branching

structure of the refined model.

Strong refinement captures the notion of evolving a model by allowing decisions to be taken regarding maybe transitions. However, in practice, model elaboration can lead also to the necessity of extending the alphabet of the system to describe behaviour aspects that previously had not been taking into account. For example, in our case study we may eventually want to describe with a lower level of abstraction how a particular function works. This can be seen in Figure 2.3 where the τ transition from state 1_{x_1} to state 0 of the initial model has been replaced by the sequence *readList*, *showList*. Once this has been done we will need a way to ensure that the new model does in fact conform the previous one. Since strong refinement requires the alphabets of the models being compared to be equal we will need to check this using the concept of hiding, where transitions with new symbols of the alphabet are replaced with tau transitions. However, this refinement does not consider tau transitions as silent ones and therefore the comparison result of these models will not be as expected. For the stated reasons, in the case of the software development processes we are studying, it is imperative to use an observational semantics, i.e. a semantics that in some way considers tau transitions as silent ones. As far as we know weak refinement [3] is the only observational semantics currently defined over MTS, and so we will consider it in the following section.

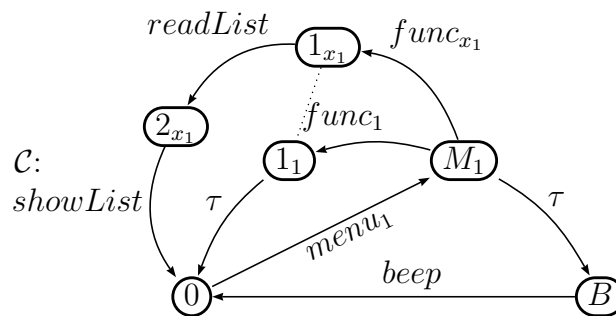


Figure 2.3: A model where the behaviour of functionality associated to $func_{x_1}$ has been detailed.

2.3 Weak Semantics

Weak MTS refinement also defined by Larsen [3] 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. Weak MTS refinement is this defined based on the definition of LTS weak bisimulation, as can be seen in the following definition.

Definition 2.3.1 (Weak Refinement). *N is a weak refinement of M , written $M \preceq_O N$, 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. $(M \xrightarrow{\ell}_r M') \Rightarrow (\exists N' \cdot N \xRightarrow{\hat{\ell}}_r N' \wedge (M', N') \in R)$
2. $(N \xrightarrow{\ell}_p N') \Rightarrow (\exists M' \cdot M \xRightarrow{\hat{\ell}}_p M' \wedge (M', N') \in R)$

It is worth noting that the relation between weak bisimulation and weak refinement follows the same pattern used to extend strong bisimulation into a refinement. Also, if a model N is a strong refinement of model M ($M \preceq N$) then N is also a weak refinement of M ($M \preceq_O N$). Finally, as with strong refinement, a notion of (weak) implementation can be defined between MTSs and LTSs.

We now consider again the model \mathcal{C} described in Figure 2.3. If we hide the new actions and then use weak refinement to compare it with the initial model \mathcal{A} , it can be stated that \mathcal{C} is a refinement of \mathcal{A} being the refinement relation $R = \{(0, 0), (B, B), (M_1, M_1), (1_1, 1_1), \dots, (1_{x_1}, 1_{x_1}), (0, 2_{x_1})\}$. Thus, as expected, with this semantics the analysed model is an adequate evolution of the initial one.

Having introduced weak semantics and shown one example of its applicability we now analyse more closely its adequacy of to support a model elaboration. Our analysis is based on one particular implementation of the MTS \mathcal{A} , which we present in Figure 2.4 and denote \mathcal{I} . Considering weak semantics, since $R = \{(0, 0), (B, B), (M_1, M_1), (1_1, 1_1), \dots, (1_{x_1}, 1_{x_1})\}$ is a refinement relation between \mathcal{A} and \mathcal{I} , the following LTS is a possible implementation of the system:

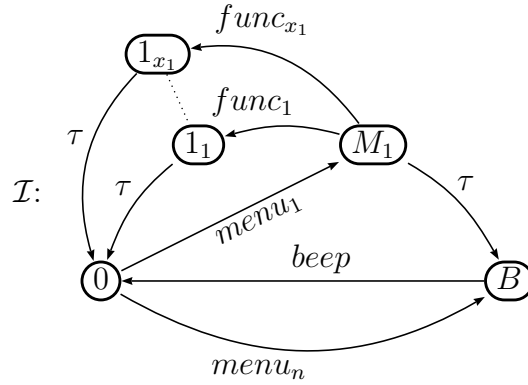


Figure 2.4: A valid implementation of the initial model according to weak refinement.

From the original model \mathcal{A} (Figure 2.1), it is not decided if $menu_n$ is to be included in the final system, but if it were 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 $func_1 \dots func_{x_n}$ 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 does not preserve the branching structure of the original model.

A more in depth analysis of this semantics allows us to gain some insight as to why weak refinement leads to such unintuitive implementations. For this we use an alternate (yet equivalent) standard definition of weak semantics. Weak semantics can be thought of as simply applying strong semantics to the models obtained from performing the transitive closure of tau transitions.

Definition 2.3.2 (Observational Graph). *Given an MTS $M = (S, L, \Delta^r, \Delta^p, s_0)$, the observational graph of M is the derived MTS $Obs(M) = (S, L, \Delta_o^r, \Delta_o^p, s_0)$ where Δ_o^r, Δ_o^p are given by:*

1. $\Delta_o^r = \{M \xrightarrow{\ell} M' \mid M \xRightarrow{\tau}_r M'\}$
2. $\Delta_o^p = \{M \xrightarrow{\ell} M' \mid M \xRightarrow{\tau}_p M'\}$

Property 2.3.3 (Weak Refinement). *N is a weak refinement of M , written*

$M \preceq_O N$, if and only if $Obs(M) \preceq Obs(N)$.

We now revisit the case study with this alternative definition of weak semantics. Consider the transitive closure of the model in Figure 2.1 depicted in Figure 2.5. The transition labelled $menu_n$ in Figure 2.1 gives rise to two different maybe transitions in Figure 2.5, i.e. $0 \xrightarrow{menu_n}_m M_n$ and $0 \xrightarrow{menu_n}_m B$. In obtaining the implementation in Figure 2.4, we can consider that these two transitions were refined inconsistently: $0 \xrightarrow{menu_n}_m B$ became a valid behaviour while $0 \xrightarrow{menu_n}_m M_n$ became forbidden. These kind of “inconsistent” decisions that weak refinement allows over different transitions in the closed model that were originated from the same maybe transition are the cause for these unexpected implementations.

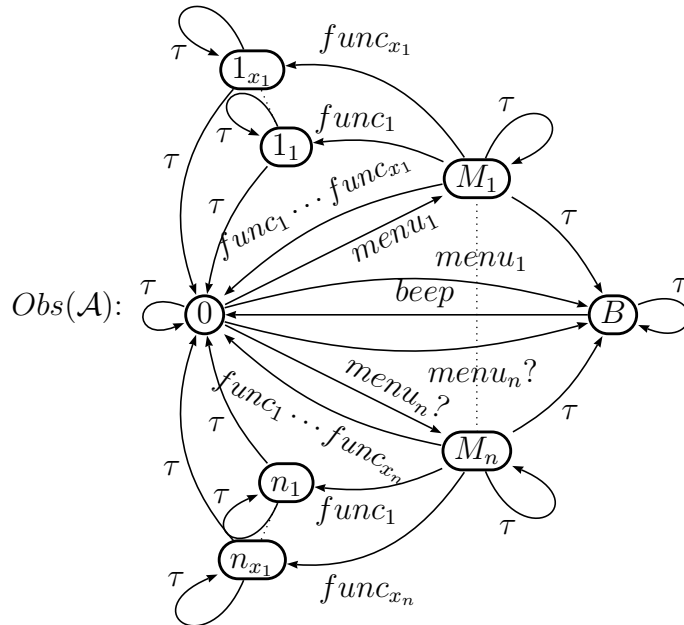


Figure 2.5: A graphical representation of the observational graph of model in Figure 2.1

Chapter 3

New Semantics

In the previous chapter we analyse the shortcomings of strong and weak semantics for our modelling purposes. The former semantics does not distinguish observable from unobservable actions and hence does not support elaboration when varying the level of abstraction of partial models. The latter does not preserve branching behaviour adequately hence allowing implementations of partial models that contradict the intuition users may have of partial models. In this chapter we first explore the intuition we have for MTS semantics by analysing specific examples and then formally define a novel semantics that captures this intuition.

3.1 Exploration

We have already settled that the new semantics should be an observational one that also preserves the branching structure. In this section we present a series of examples that will allow us to further clarify the behaviour we are looking for in the new semantics. Each of the examples consists of a model M and a model N and all of them fulfill the following condition: according to weak semantics N is considered to be a refinement of M whereas according to strong semantics no refinement relation can be defined between them. In each case we are going to state whether we expect the new semantics to define or not a refinement relation between the two models and explain the reasons that support this choice.

$$\mathcal{M}_1 : \textcircled{0} \xrightarrow{\tau?} \textcircled{1} \xrightarrow{a} \textcircled{2} \equiv \mathcal{N}_1 : \textcircled{0} \xrightarrow{a?} \textcircled{1} \quad (3.1)$$

The models in this first example are weak equivalent and since there is no branching in their structure we can state that they have the same branching structure. For the previous reasons we would like these models to be considered equivalent in the new semantics.

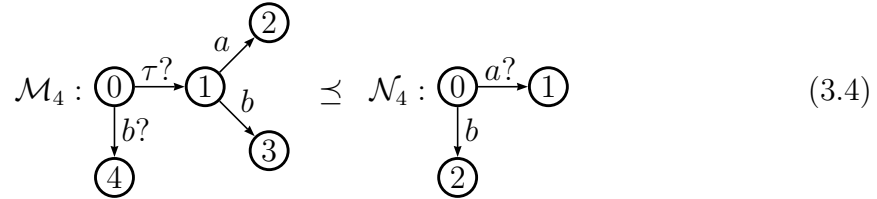
We will now add to both models a new required transition from the initial state and we will analyse the impact of this addition on the relation between the two models according to the desired behaviour for the new semantics.

$$\mathcal{M}_1 : \begin{array}{c} \textcircled{0} \xrightarrow{\tau?} \textcircled{1} \xrightarrow{a} \textcircled{2} \\ \downarrow b \\ \textcircled{3} \end{array} \not\equiv \mathcal{N}_2 : \begin{array}{c} \textcircled{0} \xrightarrow{a?} \textcircled{1} \\ \downarrow b \\ \textcircled{2} \end{array} \quad (3.2)$$

In this case it can be seen that a valid implementation of \mathcal{N}_2 is $(a+b)$. However, we do not want that implementation to be a valid one for \mathcal{M}_2 since intuitively this would not preserve the branching structure. If we look at model \mathcal{M}_2 we will realize that in order to take a we should always reach a state from where a can be taken but b cannot. Thus, allowing $(a+b)$ as a possible implementation would violate the branching structure and so in the new semantics \mathcal{N}_2 should not constitute a refinement of \mathcal{M}_2 .

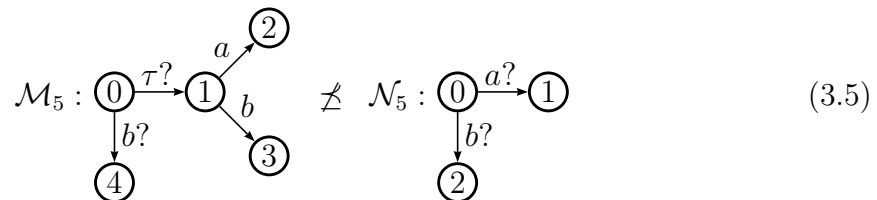
$$\mathcal{M}_3 : \begin{array}{c} \textcircled{0} \xrightarrow{\tau?} \textcircled{1} \\ \begin{array}{l} \nearrow a \textcircled{2} \\ \searrow b \textcircled{3} \end{array} \end{array} \not\equiv \mathcal{N}_3 : \begin{array}{c} \textcircled{0} \xrightarrow{a?} \textcircled{1} \\ \searrow b \textcircled{2} \end{array} \quad (3.3)$$

In example 3.3 our intention with model \mathcal{M}_3 is to describe that either the system cannot take any observable transition or it can take both a and b ($a+b$). On the other hand model \mathcal{N}_3 allows b as a possible implementation and hence we do not want \mathcal{N}_3 to be a refinement of \mathcal{M}_3 in the new semantics.



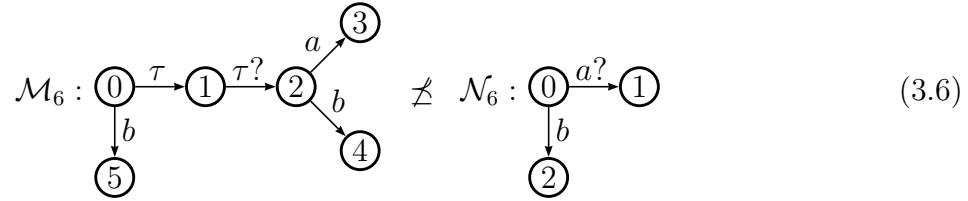
In example 3.4 model \mathcal{N}_4 allows for implementations with either one or two branches. While a one branch implementation of \mathcal{N}_4 implies transition a has been forbidden, a two branches implementation can be obtained by making a a required transition. In both cases an equivalent model can be obtained from \mathcal{M}_4 if we forbid or require its τ transition respectively and therefore we can say that the branching structure of \mathcal{M}_4 is preserved by \mathcal{N}_4 .

It should be noted that the main difference between this example and the previous one is that in this case after having taken the τ transition in model \mathcal{M}_4 one still has the possibility of taking b . Therefore, there is no such situation in which a can be taken but b cannot. To sum up, since \mathcal{N}_4 is a weak refinement of \mathcal{M}_4 and preserves its branching structure we would want our semantics to consider this relationship as a refinement.



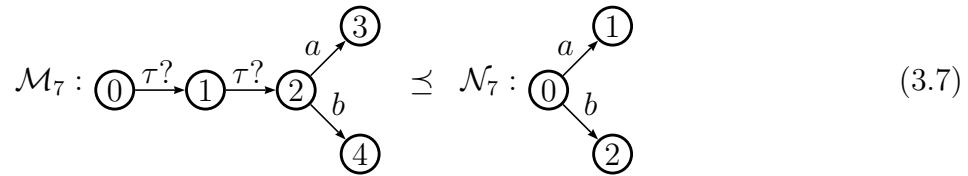
The example shown in 3.5 is different from the one shown in 3.4 in that transition b in model \mathcal{N}_5 is now a maybe transition. As a result this model

allows a as a possible implementation. However, as we have already analysed in the previous two examples, model \mathcal{M}_5 has a structure such that if transition a can be taken then transition b is also possible. For this reason we do not want the new semantics to consider model \mathcal{N}_5 to be a refinement of model \mathcal{M}_5 .

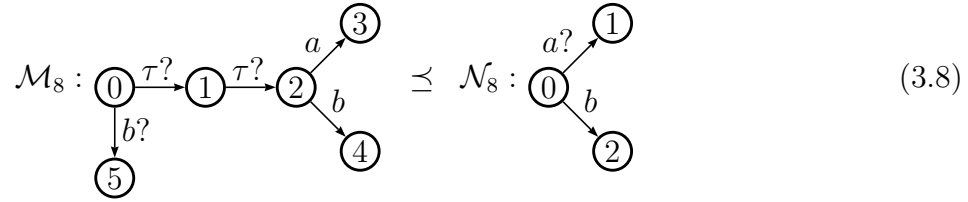


Model \mathcal{M}_6 is different from models \mathcal{M}_4 and \mathcal{M}_5 in that it adds a new required τ transition in such a way that any implementation that aims to preserve the branching structure should have two branches. Therefore, b is not a valid implementation of this model. Since \mathcal{N}_6 accepts this implementation as a valid one, we expect the new semantics to do not consider \mathcal{N}_6 as a refinement of \mathcal{M}_6 .

Let's consider now the following example:



Model \mathcal{N}_7 is a weak implementation of model \mathcal{M}_7 and since obviously the latter preserves the structure of the former we would like the new semantics to consider model \mathcal{N}_7 as an implementation of \mathcal{M}_7 .



In order to study example 3.8 we will breakdown the analysis according to the two possible set of implementations of model \mathcal{N}_8 , i.e. according to whether transition a in this model becomes required or forbidden.

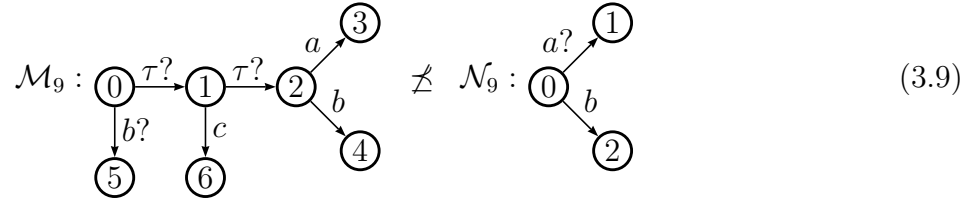
If a is forbidden then the resulting implementation of \mathcal{N}_8 is also a valid implementation of \mathcal{M}_8 . To attain this implementation of model \mathcal{M}_8 we should simply define that the maybe transition τ between the initial state and state 1 in this model becomes forbidden and the maybe transition b between the initial state and state 5 becomes required.

On the other hand, if transition a is required then model \mathcal{N}_8 becomes the same as model \mathcal{N}_7 . As already explained in the previous example, we would like this model to be considered a valid implementation of model \mathcal{M}_7 by the new semantics. But, since even for strong semantics \mathcal{M}_7 is considered to be a refinement of \mathcal{M}_8 achieved by forbidding in this latter model the b maybe transition between the initial state and state 5, then by transitivity (which we would expect to hold) we can say that this implementation of model \mathcal{N}_8 should also be a valid implementation of model \mathcal{M}_8 .

Therefore, since we expect both sets of implementations of \mathcal{N}_8 to be valid implementations of \mathcal{M}_8 we can conclude that the expected behaviour of the new semantics in this case should be to accept \mathcal{N}_8 as a refinement of \mathcal{M}_8 .

Another way to analyse this example is by realizing that example 3.8 is very similar to example 3.4, the only difference being that a maybe τ transition has been added. Since the way this transition has been added is such that the branching structure remains the same we can simply apply a similar analysis that the one performed for example 3.4 and again conclude that model \mathcal{N}_8 should be con-

sidered a refinement of model \mathcal{M}_8 according to the expected behaviour of the new semantics.



Finally, consider example 3.9 where model \mathcal{N}_9 accepts $(a+b)$ as a valid implementation. If we analyse model \mathcal{M}_9 we will see that intuitively any valid implementation of this model that includes transition a should also include the possibility of taking transition c before. This is another example that depicts how weak semantics allows for refinements that contradicts the intuitive idea of elaboration of partial models since it does not preserve branching behaviour adequately.

3.2 Definition

In this section we define a new semantics that has the desired properties of both weak and strong semantics, i.e. an observational semantics that preserve branching structure. To do this we consider a third equivalence over LTS called branching equivalence which, as we have already seen in the background, can be situated between strong and weak equivalences.

As shown previously, the three equivalences on LTS are given by a symmetric simulation relation, the difference between them is the way a transition on one model is simulated by the other model. Figure 3.1 shows a graphical representation of how a transition is simulated in each of these three equivalences. We have also shown strong and weak MTS refinement to be loosely based on the corresponding LTS equivalences by having a slightly asymmetric bisimulation in that every required transition in the less refined model must be simulated by the refined model using only required transitions, and every possible transition in the refined model must be simulated by possible transitions of the less refined model.

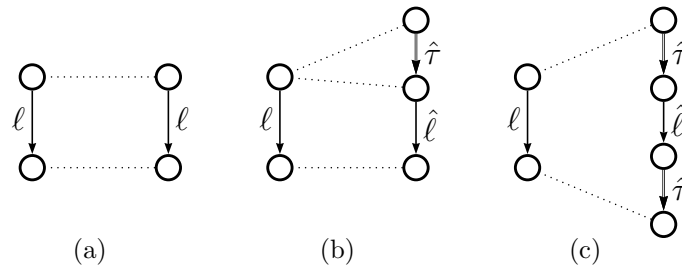


Figure 3.1: Depiction of how a transition is simulated in bisimulation: (a) strong; (b) branching; (c) weak.

Unfortunately, LTS branching bisimulation cannot be adapted in a similar way to produce an adequate MTS refinement. Consider the refinement from applying to LTS branching equivalence a similar pattern as for LTS strong and weak equivalence:

Definition 3.2.1 (Naïf branching refinement 1). *N is a branching refinement of M , written $M \preceq_{b1} N$, 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 $\ell \in Act_\tau$:*

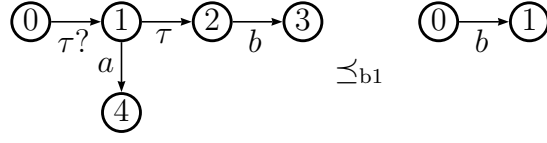


Figure 3.2: Example of a refinement according Definition 3.2.1 where the branching structure of the less refined process is not preserved.

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

The above definition does not lead to an adequate refinement notion since it does not preserve branching structure. The Figure 3.2 shows an example of a model refining another model without preserving the branching structure; $R = \{(0, 0), (2, 0), (3, 1)\}$ is the relation that relates these models according to the previous definition.

The reason why this definition does not preserve the branching structure is that it does not guaranty that all intermediate states of $M \xrightarrow{\tau}_p M'$ are related to N , as the stuttering lemma states for branching equivalence. To amend this problem one could think it might be worth reinforcing this definition by explicitly requiring stuttering to be preserved. The definition that would be obtained is the following:

Definition 3.2.2 (Naïf 2 branching refinement). *N is a branching refinement of M , written $M \preceq_{b2} N$, 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. $(M \xrightarrow{\ell}_r M') \Rightarrow (\exists N_0, \dots, N_n, N' \cdot$
 $N_i \xrightarrow{\tau}_r N_{i+1} \forall 0 \leq i < n \wedge N_n \xrightarrow{\hat{\ell}}_r N' \wedge$
 $N_0 = N \wedge (M, N_i) \in R \forall 0 \leq i \leq n \wedge (M', N') \in R)$
2. $(N \xrightarrow{\ell}_p N') \Rightarrow (\exists M_0, \dots, M_n, M' \cdot$
 $M_i \xrightarrow{\tau}_p M_{i+1} \forall 0 \leq i < n \wedge M_n \xrightarrow{\hat{\ell}}_p M' \wedge$
 $M_0 = M \wedge (M_i, N) \in R \forall 0 \leq i \leq n \wedge (M', N') \in R)$

This definition preserves branching structure since every intermediate state a model goes through when simulating a transition on the other model is actually related to the initial state of that transition. Intuitively, this means that none of those intermediate states present more or less behaviour than the initial state. In particular it solves the problem of Figure 3.2

However, this definition has another problem. If we consider the set of implementations of M and N the fact that one is included in the other one does not imply the existence of a refinement relation between them, i.e. $\mathcal{I}_{b2}[M] \supseteq \mathcal{I}_{b2}[N] \not\Rightarrow M \preceq_{b2} N$. Figure 3.3 shows an example of this case. While the two models shown have the same set of possible implementations according to definition 3.2.2, there is no appropriate refinement relation for $A2 \preceq_{b2} A1$. Because of this, definition 3.2.2 is inappropriate since one could end up creating a model that properly restricts the set of possible implementations defined by the previous model but that could be not considered a valid evolution due to the inexistence of a refinement relation between the two models.

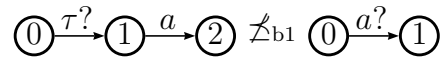


Figure 3.3: Two models that have the same set of implementations according Definition 3.2.2 but one is not a refinement of the other according the same definition.

To overcome these limitations, and recalling that a MTS semantics is completely defined by stating which are valid implementations for a model, we adapt branching equivalence into a 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 .

Definition 3.2.3 (Branching Implementation). *Let \wp be the universe of all LTS and δ of all MTS. Let M a MTS, and I a LTS such that $\alpha M = \alpha I$, I is a branching implementation of M , written $M \preceq_b I$, if and only if (M, I) is contained in some implementation relation $R \subseteq \delta \times \wp$ for which the following holds for all*

$\ell \in Act_\tau$:

1. $(M \xrightarrow{\ell}_r M') \Rightarrow (\exists I_0, \dots, I_n, I' \cdot$
 $I_i \xrightarrow{\tau} I_{i+1} \forall 0 \leq i < n \wedge I_n \xrightarrow{\hat{\ell}} I' \wedge$
 $I_0 = I \wedge (M, I_i) \in R \forall 0 \leq i \leq n \wedge (M', I') \in R)$
2. $(I \xrightarrow{\ell} I') \Rightarrow (\exists M_0, \dots, M_n, M' \cdot$
 $M_i \xrightarrow{\tau}_p M_{i+1} \forall 0 \leq i < n \wedge M_n \xrightarrow{\hat{\ell}}_p M' \wedge$
 $M_0 = M \wedge (M_i, I) \in R \forall 0 \leq i \leq n \wedge (M', I') \in R)$

Clearly it can be observed that if this relation is restricted to LTS it coincides with branching equivalence. It can also be easily proved that if $M \preceq_b I$ and $I \approx_b I'$ then $M \preceq_b I'$, and so this novel implementation relation is a sound extension of branching equivalence.

In this way we have defined a new semantics over MTS that extends branching equivalence.

Chapter 4

Tool Support

In the previous chapters we analysed weak and strong semantics for MTS and presented a new semantics for these models. Those three semantics determine operationally the conformity between an MTS and its implementations. This is achieved by checking the existence of a relation between the states of both models that satisfies certain bisimulation. Given two models, the task of manually determining if an implementation relation between them exists can be laborious and error-prone. Moreover, for models with more than a few states it becomes almost impracticable. For this reason we have developed a software tool that allows the user to verify if a given LTS is an implementation of a certain MTS according to the previous three semantics. Furthermore, this tool can also be used to check if a refinement relation exists between two MTS models according to weak, strong or naïf 2 branching semantics. The algorithm used to calculate the different relations is a fixed point algorithm that starts with the Cartesian product of the states and iteratively eliminates from it the pairs that are not valid according to the required bisimulation. In this chapter we introduce a tool we have developed¹ along with the fixed point algorithm used.

¹It can be downloaded from <http://www.doc.ic.ac.uk/~fdario/MTSTool/MTSChecker.zip>

4.1 Fix Point Algorithm

In this section we give a formal definition of a function over the domain of binary relations between models and we demonstrate that by applying this function iteratively it converges. Moreover, we demonstrate that if a branching implementation relation exists between models then the fix point to which the function converges is actually a branching implementation relation.

Definition 4.1.1. Let \wp be the universe of all LTS, δ of all MTS. $\mathcal{F} : \mathcal{P}(\delta \times \wp) \rightarrow \mathcal{P}(\delta \times \wp)$ is defined by:

$$\mathcal{F}(R) = \{ \langle M, I \rangle \in R \mid \forall (\ell, M') ((M \xrightarrow{\ell}_r M') \Rightarrow FS(M, M', I, \ell, R)) \wedge \forall (\ell, I') ((I \xrightarrow{\ell} I') \Rightarrow BS(M, I, I', \ell, R)) \}$$

where:

$$\begin{aligned} FS(M, M', I, \ell, R) &\equiv \exists (I_0, \dots, I_n, I') (I_i \xrightarrow{\tau} I_{i+1} \forall 0 \leq i < n \wedge I_n \xrightarrow{\hat{\ell}} I' \wedge \\ &\quad \wedge I_0 = I \wedge (M, I_i) \in R \forall 0 \leq i \leq n \wedge (M', I') \in R) \\ BS(M, I, I', \ell, R) &\equiv \exists (M_0, \dots, M_n, M') (M_i \xrightarrow{\tau}_p M_{i+1} \forall 0 \leq i < n \wedge M_n \xrightarrow{\hat{\ell}}_p M' \wedge \\ &\quad \wedge M_0 = M \wedge (M_i, I) \in R \forall 0 \leq i \leq n \wedge (M', I') \in R) \end{aligned}$$

Theorem 4.1.2. Let be $R, P \subseteq \delta \times \wp$ such as R is branching implementation relation. If $R \subseteq P$ then $R \subseteq \mathcal{F}(P)$

Definition 4.1.3. Let A be an LTS (MTS) we define the “reachability” set of A as $Reach(A) = \{ A' \mid \exists w \in Act^*_\tau \cdot A \xrightarrow{w} A' \}$

Remark 4.1.4. $Reach(A)$ is a finite set by definition of LTS (MTS).

Theorem 4.1.5. Let M be an MTS and I an LTS, and $R_0, R_1, \dots \subseteq \delta \times \wp$ given by $R_0 = Reach(M) \times Reach(I)$ and $R_{i+1} = \mathcal{F}(R_i)$. Then there exists j such that:

1. $R_j = R_i \forall i > j$

2. $\langle M, I \rangle \in R_j$ iff $M \preceq_b I$

Proof.

1. On the one hand, the sequence $(R)_i$ is monotonically decreasing since by definition of \mathcal{F} we know that $R_{i+1} \subseteq R_i$ for every i . On the other hand, $Reach(M)$ and $Reach(I)$ are finite and consequently the relation R_0 is finite. Therefore, there exists j from which the sequence converges. Note that j is less than or equal to the cardinality of R_0 .
2. \Rightarrow) By definition of sequence $(R)_i$ we know that $R_{j+1} = \mathcal{F}(R_j)$. Based on 1. we also know that $R_{j+1} = R_j$. Consequently, $R_{j+1} = \mathcal{F}(R_{j+1})$, which is equivalent to:

$$R_{j+1} = \{ \langle M, I \rangle \in R_{j+1} \mid \forall (\ell, M') ((M \xrightarrow{\ell}_r M') \Rightarrow FS(M, M', I, \ell, R_{j+1})) \wedge \forall (\ell, I') ((I \xrightarrow{\ell} I') \Rightarrow BS(M, I, I', \ell, R_{j+1})) \}$$

Since FS and BS correspond to conditions 1 and 2 in Definition 3.2.3, respectively, then R_{j+1} is a branching implementation relation. Finally, if $\langle M, I \rangle \in R_j = R_{j+1}$ then $M \preceq_b I$

\Leftarrow) If $M \preceq_b I$ then a branching implementation relation R such that $\langle M, I \rangle \in R$ exists. Without loss of generality we can assume that $R \subseteq Reach(M) \times Reach(I) = R_0$. Therefore, based on Theorem 4.1.2 we can say that $R \subseteq R_i$ for every i . In particular $R \subseteq R_j$ and since $\langle M, I \rangle \in R$ we conclude that $\langle M, I \rangle \in R_j$

□

Proof. theorem 4.1.2

If $R \not\subseteq \mathcal{F}(P)$ then there exists $\langle M, I \rangle \in R$ such that $\langle M, I \rangle \notin \mathcal{F}(P)$. By definition of \mathcal{F} at least one of the following predicates must be false:

1. $\forall(\ell, M')((M \xrightarrow{\ell}_r M') \Rightarrow FS(M, M', I, \ell, R))$
2. $\forall(\ell, I')((I \xrightarrow{\ell} I') \Rightarrow BS(M, I, I', \ell, R))$

Suppose predicate 1 is false. Then there exists ℓ, M' such that $M \xrightarrow{\ell}_r M'$ holds and $FS(M, M', I, \ell, R)$ is false. Since $\langle M, I \rangle \in R$ and $M \xrightarrow{\ell}_r M'$ holds, using Definition 3.2.3 we get that: $\exists(I_0, \dots, I_n, I')(I_i \xrightarrow{\tau} I_{i+1} \forall 0 \leq i < n \wedge I_n \xrightarrow{\hat{\ell}} I' \wedge I_0 = I \wedge (M, I_i) \in R \forall 0 \leq i \leq n \wedge (M', I') \in R)$ holds. But this expression is exactly $FS(M, M', I, \ell, R)$ hence we reach an absurd, so predicate 1 must be true. Using the same reasoning we get that predicate 2 must be true, and consequently we prove that $R \subseteq \mathcal{F}(P)$.

□

At this point we have proved the feasibility of using a fixed point algorithm to obtain a branching implementation relation. This can be proved in a similar way also for the other bisimulations discussed in the previous chapters. For the particular case of LTS equivalence relations more efficient algorithms are known [10, 11]. However, this is not the case for implementation or refinement relations.

4.2 Implementation

The software tool has been implemented using Java 5.

The main objective of the development was to generate a tool that allows the user to compare the relation between models according to different semantics. Since it is meant to be a research tool that contributes to the assessment of semantics' definitions, the emphasis was on flexibility to include new semantics as required rather than being on performance. The key algorithm implemented is a fixed point algorithm used to calculate the different bisimulation relations. This algorithm receives as a parameter the bisimulation rules that should be considered. In order to express the complexity of this algorithm for the different

semantics we will denote the amount of states of the largest model as n . Then, the spacial complexity is $O(n^2)$ for any semantics while the time complexity is in the worst case $O(n^5 * \log(n))$ for strong, $O(n^6 * \log(n))$ for weak, and $O(n^4 * n! * \log(n))$ for branching semantics.

The main additional algorithms that have been implemented and their corresponding complexity are the following:

- An algorithm that calculates the closure by τ transitions for weak bisimulation, its complexity being polynomial.
- A DFS algorithm that finds all paths of the form $(\xrightarrow{\tau})^* \xrightarrow{\hat{\ell}}$ for branching bisimulation. In this case, the worst case complexity is factorial since the quantity of those paths is bounded by the factorial of the number of states.

In order to represent the models in such a way that could be interpreted by the software tool, a language called FSP has been adopted. This language has a well defined semantics for LTS and is used by Labelled Transition System Analyser (LTSA) tool [12]. In the following section we introduce this language in more detail and explain how it has been extended to also express MTS.

4.2.1 FSP

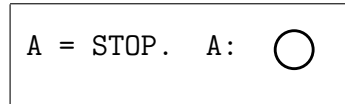
FSP stands for Finite State Processes. It is a simple process algebra notation with a semantics in terms of LTS introduced by Magee et al [12]. It has been designed to textually specify LTS in a concise way.

A process written in FSP is given by an expression consisting of composition operators, processes and actions. While process' names begin with an uppercase letter, actions' names start with a lowercase letter. A process is defined by one or more local processes separated by commas, and the end of the definition is marked with a full stop.

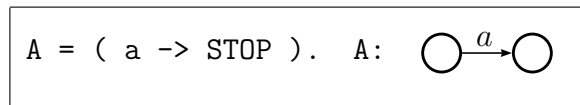
We will now define the basic operators using the following notation: x and y denote actions while P and Q denote processes. In addition, we include an

example for each of the operators. For a full description of FSP syntax and semantics refer to [12].

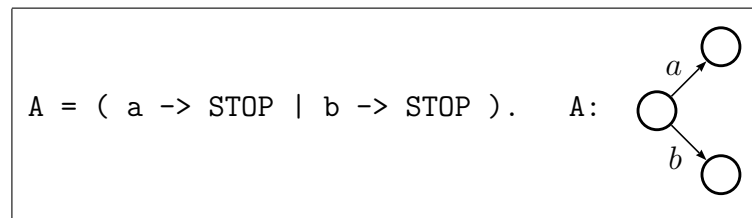
- **Primitive Process ‘STOP’**: FSP has the primitive local process **STOP** that is a process that cannot engage in any action.



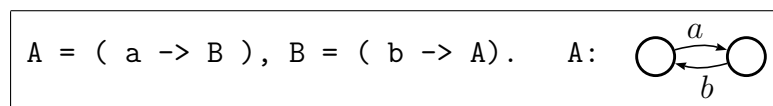
- **Action Prefix ‘ \rightarrow ’**: $(x \rightarrow P)$ describes a process that initially engages in the action x and then behaves exactly as described by P .



- **Choice ‘ $|$ ’**: $(x \rightarrow P \mid y \rightarrow Q)$ describes a process which initially engages in either action x or y . If the first action is taken then the subsequent behaviour is described by P while if the second action is taken Q describes the subsequent behaviour.



- **Recursion**: the behaviour of a process can be defined recursively. The recursion may be directly in terms of the process being defined, or indirectly in terms of other processes.

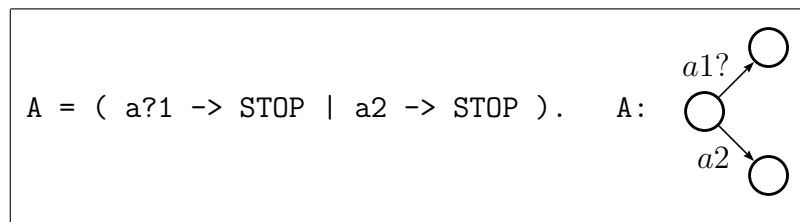


In order to be able to define also MTSs we have developed an extension to FSP. The idea is to allow the use of question marks in such a way that if

an action's name includes at least one question mark then the corresponding transition should be interpreted as a maybe one. Otherwise, the action represents a required transition.

The events of the MTS model are obtained by removing from the action's name all occurrences of the question mark symbol. For example, (`readLevel? -> STOP`) represents a maybe transition through the `readLevel` event and can also be written using the following notation: (`read?Level -> STOP`). The decision of allowing the inclusion of question marks in any part of the action's name apart from the first symbol is due to the fact that some advanced FSP operators generate actions with suffixes. Therefore, it is not possible to guarantee that if a question mark is included in the name it will be its last symbol.

Following there is an example of a simple MTS described by an FSP expression.



4.2.2 User Guide

The interaction with the software tool is through the command line. In order to invoke the application it is necessary to pass as arguments the name of the file where all models are defined and the name of the file that states the pairs of those models that have to be checked. The following is an example of the syntax used to invoke the tool:

Figure 4.1: `userGuide/commandLine`

```
java -jar MTSChecker.jar example.fsp example.queries
```

The models are defined in a text file using FSP notation. These files can be created either with any text editor or by using the LTSA version provided along

with the tool. This version has been specially modified in order to handle the extended FSP syntax which allows for question marks.

Figure 4.2: Example of an FSP input file.

```
M_02 = ( b? -> STOP | _tau? -> a -> STOP ).
N_02 = ( b -> STOP | a? -> STOP ).
I_02_01 = ( a -> STOP | b -> STOP ).
```

The queries file is also a text file but with the following format: each line consists of the names of two models separated by a space. The system interprets each of those lines as a query to check if the second model is a refinement or an implementation of the first one depending on the former model being an MTS or an LTS respectively. In either case the software tool tests the relation between those models according to the applicable semantics as already described. In order to allow for comments the system ignores every line starting with a % character.

Figure 4.3: Example of a queries input files.

```
%*****
% Example 2
%*****
M_02 N_02

M_02 I_02_01
N_02 I_02_01
```

Finally, we show the output obtained when running the application using the example input files.

Figure 4.4: Output for the example input files.

```
Connected Compiler Factory
Compiled: N_02
Compiled: I_02_01
Compiled: M_02

M_02 - N_02 checking for refinement
Strong           : false  time: 59 (mseg)
```

Naive Branching	:	false	time: 18 (mseg)
Weak	:	true	time: 3 (mseg)
M_02 - I_02_01 checking for implementation			
Strong	:	false	time: 8 (mseg)
Branching	:	false	time: 1 (mseg)
Weak	:	true	time: 2 (mseg)
N_02 - I_02_01 checking for implementation			
Strong	:	true	time: 0 (mseg)
Branching	:	true	time: 5 (mseg)
Weak	:	true	time: 0 (mseg)

Chapter 5

Validation

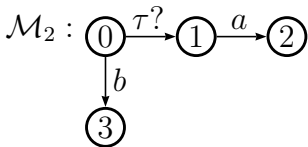
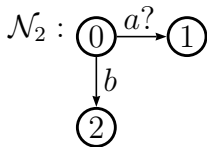
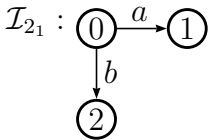
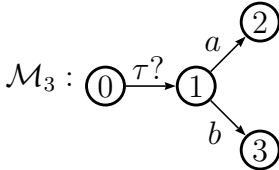
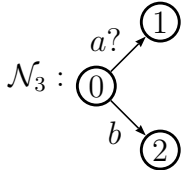
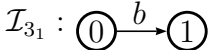
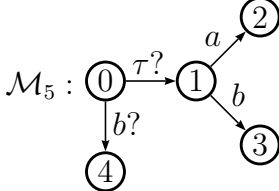
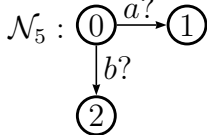
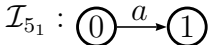
In section 3.1 we explored the behaviour desired for a new MTS semantics based on the analysis of a series of examples. Furthermore, in section 3.2 we formally defined a new MTS semantics by providing an implementation notion. In this chapter we validate that semantics by assessing if it complies with the expected behaviour for the examples analysed in section 3.1 and for the case study. In order to do so we should bear in mind that according to the definition of branching semantics \mathcal{N} is a refinement of \mathcal{M} iff every implementation of \mathcal{N} is also an implementation of \mathcal{M} . Since the set of possible implementations is infinite, the strategy to validate the definition will be the following:

- Firstly, we examine the results achieved for all the examples for which intuitively we do not consider model \mathcal{N}_i to be a refinement of model \mathcal{M}_i , i.e. $\mathcal{M}_i \preceq_{\text{O}} \mathcal{N}_i$ but $\mathcal{M}_i \not\preceq_{\text{b}} \mathcal{N}_i$. We demonstrate that the defined semantics matches the expected behaviour by showing a counterexample consisting of a model \mathcal{I}_i that is actually an implementation of model \mathcal{N}_i but not of \mathcal{M}_i according to the newly defined branching semantics.
- Secondly, we study the examples we do consider to be valid refinements, i.e. $\mathcal{M}_i \preceq_{\text{b}} \mathcal{N}_i$. Due to the infinite set of possible implementations, in order to validate that every implementation of \mathcal{N}_i is also an implementation of \mathcal{M}_i , we analyse the results for at least one member of each of the equivalence classes of the implementations set of \mathcal{N}_i given by branching equivalence.

To further complete the validation we have also included for each example an LTS model that is not a valid branching implementation of \mathcal{M}_i and therefore it should not be a valid branching implementation of \mathcal{N}_i either.

All these tests¹ have been performed using the software tool already described in the previous chapter. The results are shown in the following tables, where True (False) in a cell indicates whether the model in that row is (is not) a branching implementation of the model in the corresponding column.

Table 5.1: Validating that \mathcal{N} do not refine \mathcal{M}

	\mathcal{M}_2 : 	\mathcal{N}_2 : 
\mathcal{I}_{2_1} : 	False	True
	\mathcal{M}_3 : 	\mathcal{N}_3 : 
\mathcal{I}_{3_1} : 	False	True
	\mathcal{M}_5 : 	\mathcal{N}_5 : 
\mathcal{I}_{5_1} : 	False	True

¹These tests are defined in exploration.fsp and exploration.queries files which can be downloaded from <http://www.doc.ic.ac.uk/~fdario/MTSTool/>

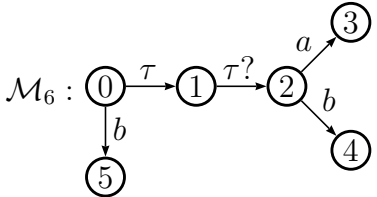
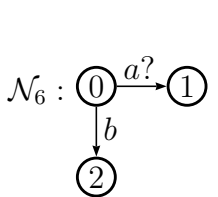
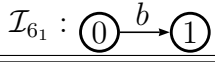
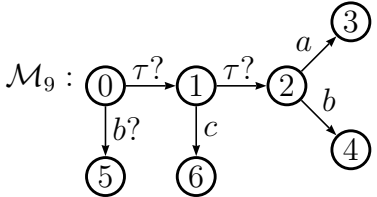
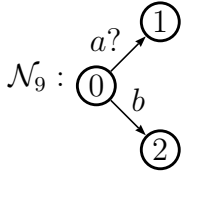
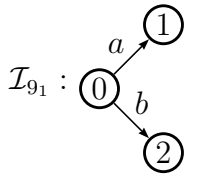
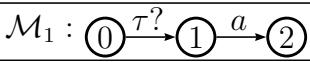
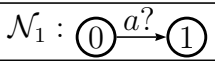
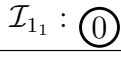
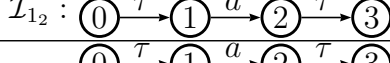
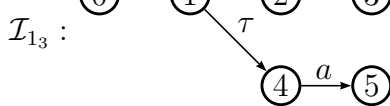
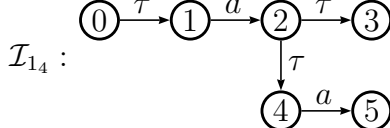
	\mathcal{M}_6 : 	\mathcal{N}_6 : 
\mathcal{I}_{6_1} : 	False	True
	\mathcal{M}_9 : 	\mathcal{N}_9 : 
\mathcal{I}_{9_1} : 	False	True

Table 5.2: Validating that \mathcal{N} refines \mathcal{M}

	\mathcal{M}_1 : 	\mathcal{N}_1 : 
\mathcal{I}_{1_1} : 	True	True
\mathcal{I}_{1_2} : 	True	True
\mathcal{I}_{1_3} : 	True	True
\mathcal{I}_{1_4} : 	False	False

	$\mathcal{M}_4 : \begin{array}{c} \textcircled{0} \xrightarrow{\tau?} \textcircled{1} \xrightarrow{a} \textcircled{2} \\ \textcircled{0} \xrightarrow{b?} \textcircled{4} \\ \textcircled{1} \xrightarrow{b} \textcircled{3} \end{array}$	$\mathcal{N}_4 : \begin{array}{c} \textcircled{0} \xrightarrow{a?} \textcircled{1} \\ \textcircled{0} \xrightarrow{b} \textcircled{2} \end{array}$
$\mathcal{I}_{4_1} : \begin{array}{c} \textcircled{0} \\ \downarrow b \\ \textcircled{2} \end{array}$	True	True
$\mathcal{I}_{4_2} : \begin{array}{c} \textcircled{0} \xrightarrow{a} \textcircled{1} \\ \downarrow b \\ \textcircled{2} \end{array}$	True	True
$\mathcal{I}_{4_3} : \begin{array}{c} \textcircled{0} \xrightarrow{\tau} \textcircled{1} \xrightarrow{a} \textcircled{2} \\ \downarrow b \\ \textcircled{3} \end{array}$	False	False
	$\mathcal{M}_7 : \begin{array}{c} \textcircled{0} \xrightarrow{\tau?} \textcircled{1} \xrightarrow{\tau?} \textcircled{2} \xrightarrow{a} \textcircled{3} \\ \textcircled{2} \xrightarrow{b} \textcircled{4} \end{array}$	$\mathcal{N}_7 : \begin{array}{c} \textcircled{0} \xrightarrow{a} \textcircled{1} \\ \textcircled{0} \xrightarrow{b} \textcircled{2} \end{array}$
$\mathcal{I}_{7_1} : \begin{array}{c} \textcircled{0} \xrightarrow{a} \textcircled{1} \\ \textcircled{0} \xrightarrow{b} \textcircled{2} \end{array}$	True	True
$\mathcal{I}_{7_2} : \begin{array}{c} \textcircled{0} \xrightarrow{\tau} \textcircled{1} \xrightarrow{a} \textcircled{2} \\ \textcircled{1} \xrightarrow{b} \textcircled{3} \end{array}$	True	True
$\mathcal{I}_{7_3} : \textcircled{0} \xrightarrow{a} \textcircled{1}$	False	False

All the above results show that the new semantics defined matches its expected behaviour. We have also validated the new semantics using the case study testing if it rejects the undesired implementation depicted in Figure 2.4 as a valid implementation of the initial model shown in Figure 2.1. The result was that branching refinement rejects that implementation as we expected.

Chapter 6

Conclusions

6.1 Summary of Contributions

In this work we have analysed from an incremental software modelling point of view the applicability of the existing implementation notions for MTS. Based on our analysis we have concluded that from this perspective these implementation notions are not adequate.

To address the limitations of existing semantics, we have studied the required characteristics for a satisfactory semantics and formally defined a new semantics for MTS with the desired characteristics given by a novel notion of implementation.

The new semantics has a correspondence with branching bisimulation[4] over LTS in a similar way Larsen's semantics informally correspond to strong and weak bisimulation over LTS. We have shown that the *natural* generalization of the branching bisimulation does not lead to an appropriate notion of refinement over MTS.

Finally, we have developed a software tool to verify refinement and implementation notions analysed in this work. We have extended FSP language adding to it the notion of maybe and required transition in order to describe MTS.

6.2 Related Work

A number of formalisms exist which allow explicit modelling of lack of information. Partial Kripke structures [13] and Kripke Modal Transition Systems [14] extend Kripke structures to support propositions in states to be one of three values (true, false, and *unknown*). In our work, states in themselves do not have any semantics, we focus only on observable system behaviour as described by the labelled transitions between states, hence we build on models in the labelled transition systems [15] style.

Our definition of Modal Transition Systems is essentially that proposed by Larsen et al. [16]. However, in [16] all MTS have the same alphabet, the universe of all labels, while we extend the definition of MTSs to include a communication alphabet in line with [12]. Making the communication alphabet allows scoping models and capturing the fact that system components may control and monitor different sets of events [17].

Numerous extensions of MTS exist such as Mixed Transition Systems [18] and disjunctive modal transition systems [19]. The semantics we proposed 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 MTSs. A slightly different approach to modelling unknown behaviour is taken in [20, 21]. In [20] Partial Labelled Transition Systems, each state is associated with a set of actions that are explicitly proscribed from happening. Extended Transition Systems [21] 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.

Related work regarding refinement and simulation has been discussed extensively throughout the work. 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 [7], testing [8] and action-refinement [22]) 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 [2] and weak refinement [3] over MTS.

6.3 Future work

Our research objective is the development of practical and effective techniques with tool support for the elaboration of comprehensive behaviour models in a context where total information is not available, such as iterative and incremental software development processes. As already discussed, MTS can be used to facilitate the elaboration of such models. In particular, we believe that they can provide the rigorous underpinning descriptions require in order to help analyse, correct, elaborate and refine behaviour specifications.

Although MTS have been studied extensively they have yet to be studied in the context of model elaboration and more generally in software engineering. In this work we have analysed the existing semantics and define a new semantics we believe is adequate for our purposes. This lays the foundations for our long term research goal.

The current definition allows us to characterize the refinement notion between models in a declarative way. Consequently, the derivation of an algorithm for refinement verification is a difficult task since it would imply obtaining all the implementations of one model and checking their conformity against the other one. Being our intention the development of a tool set that supports MTS analysis, having such algorithm becomes indispensable. In order to do so in the near future we expect to obtain an operational definition for branching refinement which will allow us to start working towards the design of the desired algorithm.

On the other hand, we expect to work on improving the efficiency of the algorithm for verifying branching implementation. We believe that there exists a polynomial solution for this problem. As a first step in this direction will be to research the possibility of extending the existing polynomial algorithms for checking branching bisimulation between LTS [10, 23]. In addition, the other

topic we expect to research in this direction is the development of algorithms that provide diagnostic information to the modeller when an implementation does not conform a partial model. This information will help the modeller to understand and correct the implementation (or the partial model).

Finally, we intend to apply the ideas discussed in this work to a real case study in order to validate them and gain feedback to drive future research.

Bibliography

- [1] Edmund M. Clarke and Jeannette M. Wing. Formal methods: state of the art and future directions. *ACM Comput. Surv.*, 28(4):626–643, 1996.
- [2] Kim Guldstrand Larsen and Bent Thomsen. A modal process logic. In *LICS*, pages 203–210. IEEE Computer Society, 1988.
- [3] Hans Hüttel and Kim Guldstrand Larsen. The use of static constructs in a modal process logic. In Albert R. Meyer and Michael A. Taitlin, editors, *Logic at Botik*, volume 363 of *Lecture Notes in Computer Science*, pages 163–180. Springer, 1989.
- [4] Rob J. van Gabbeek and W. Peter Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.
- [5] Rob J. van Glabbeek. What is branching time semantics and why to use it? pages 469–479, 2001.
- [6] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.
- [7] Steve Schneider and S. A. Schneider. *Concurrent and Real Time Systems: The CSP Approach*. John Wiley & Sons, Inc., New York, NY, USA, 1999.
- [8] Rance Cleaveland and Matthew Hennessy. Testing equivalence as a bisimulation equivalence. *Formal Asp. Comput.*, 5(1):1–20, 1993.
- [9] Michael Huth. Refinement is complete for implementations. *Formal Asp. Comput.*, 17(2):113–137, 2005.

- [10] Jan Friso Groote and Frits W. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In Mike Paterson, editor, *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 626–638. Springer, 1990.
- [11] Agostino Dovier, Carla Piazza, and Alberto Policriti. A fast bisimulation algorithm. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 79–90. Springer, 2001.
- [12] Jeff Magee and Jeff Kramer. *Concurrency: state models & Java programs*. John Wiley & Sons, Inc., New York, NY, USA, 1999.
- [13] G. Bruns and P. Godefroid. “Model Checking Partial State Spaces with 3-Valued Temporal Logics”. In *CAV’99*, volume 1633 of *LNCS*, pages 274–287, 1999.
- [14] M. Huth. “Model-Checking Modal Transition Systems Using Kripke Structures”. In *Proceedings of Third International Workshop on Verification, Model-Checking, and Abstract Interpretation*, Venice, Italy, January 2002.
- [15] Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976.
- [16] K.G. Larsen and B. Thomsen. “A Modal Process Logic”. In *LICS’88*, pages 203–210, 1988.
- [17] Michael Jackson. *Software requirements & specifications: a lexicon of practice, principles and prejudices*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.
- [18] Dennis Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Eindhoven University of Technology, The Netherlands, July 1996.
- [19] K. Larsen and L. Xinxin. “Equation Solving Using Modal Transition Systems”. In *5th Annual IEEE Symposium on Logic in Computer Science*, pages 108–117, 1990.

- [20] S. Uchitel, J. Kramer, and J. Magee. “Behaviour Model Elaboration using Partial Labelled Transition Systems”. In *ESEC/FSE’03*, pages 19–27, 2003.
- [21] Robin 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.
- [22] R. Gorrieri and A. Rensink. Action refinement. Technical report, 1999.
- [23] Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Characterizing finite kripke structures in propositional temporal logic. *Theor. Comput. Sci.*, 59:115–131, 1988.